# פרק 7 – ביצוע-חוזר

עד כה הכרנו בעיות אשר לשם פתרונן ביצענו מספר תת-משימות שונות, באופן סדרתי. כלומר כל תת-משימה בסדרה בוצעה פעם אחת (ואם זו משימה שביצועה תלוי בתנאי, ייתכן שלא בוצעה אפילו פעם אחת). אולם יש בעיות אשר לצורך פתרונן יש לבצע תת-משימה אחת, או כמה תת-משימות, יותר מפעם אחת, ואולי אף מספר רב של פעמים. בפרק זה נכיר אלגוריתמים אשר מורים על חזרה שוב ושוב על ביצוע של תת-משימה (או תת-משימות). אלגוריתמים אלה כוללים **הוראה לביצוע-חוזר** של קבוצת הוראות.

בסעיף 7.1 נכיר אלגוריתמים שמבנה הביצוע-החוזר בהם הוא פשוט. באלגוריתמים אלה מספר הפעמים של הביצוע-החוזר נקבע **לפני** תחילת ביצועו.

בסעיף 7.4 נכיר אלגוריתמים שמבנה הביצוע-החוזר בהם מורכב יותר. באלגוריתמים אלה מספר הפעמים של הביצוע-החוזר לא נקבע מראש, אלא תלוי בתנאי אשר נבדק שוב ושוב במהלך הביצוע-החוזר.

## 7.1 ביצוע-חוזר מספר פעמים ידוע מראש

### מצ'ה 1

**מטרת הבעיה ופתרונה:** הצגת אלגוריתם הכולל הוראה לביצוע-חוזר מספר פעמים ידוע מראש.

> עלינו להמיר רשימת מחירים מייצוג בדולרים לייצוג בשקלים. פתחו וישמו אלגוריתם אשר הקלט שלו הוא שער ההמרה מדולרים לשקלים ואחריו רשימה של עשרה מחירים הנתונים בדולרים. הפלט שלו הוא הערך בשקלים של כל אחד מעשרת המחירים. הפלט עבור כל מחיר צריך להינתן מיד אחרי קליטתו ולפני קליטת המחיר הבא.

### פירוק הבעיה לתת-משימות

1. קליטת שער ההמרה מדולרים לשקלים
2. קליטת כל אחד מעשרת המחירים בדולרים, חישוב ערכו בשקלים והצגת הערך המחושב

? ניתן לפרט את התת-משימה השנייה. כיצד?

התת-משימה השנייה מורכבת בעצם מביצוע-חוזר, עשר פעמים, של התת-משימות הבאות:

2.1 קליטת מחיר בדולרים
2.2 חישוב ערכו של המחיר בשקלים
2.3 הצגה של הערך המחושב

דרך אחת להורות על ביצוע החוזר עשר פעמים על התת-משימות שניסחנו, היא כמובן לכתוב אותן עשר פעמים, כך:

2.1. קליטת מחיר בדולרים
2.2. חישוב ערכו של המחיר בשקלים
2.3. הצגה של הערך המחושב
2.4. קליטת מחיר בדולרים
2.5. חישוב ערכו של המחיר בשקלים
2.6. הצגה של הערך המחושב

.
.
.

2.28. קליטת מחיר בדולרים
2.29. חישוב ערכו של המחיר בשקלים
2.30. הצגה של הערך המחושב

זהו ניסוח מסורבל כמובן. עבור רשימה של עשרה מחירים נכתבות שלושים הוראות. עבור רשימה של מאה מחירים ייכתבו 300 הוראות. ובעצם, עבור רשימות מחירים באורכים שונים ייכתבו אלגוריתמים שבהם מספר הוראות שונה. כלומר, לא רק שמדובר באלגוריתמים ארוכים מאוד, אלא שעבור שינוי קטן בהגדרת הבעיה (מספר המחירים), יש צורך לבצע שינוי משמעותי באלגוריתם.

האם ניתן להימנע מן הסרבול המתואר? האם ניתן לכתוב אלגוריתם שבו יהיה אותו מספר הוראות עבור רשימות מחירים באורכים שונים?

אכן ניתן באמצעות הוראה לביצוע-חוזר לקבוצת הוראות. בפתרון הבעיה הנוכחית שבה יש להמיר עשרה מחירים ניתן להשתמש בהוראה הבאה לביצוע-חוזר:

*בצע 10 פעמים:*
*קלוט מחיר בדולרים*
*חשב את ערכו של המחיר בשקלים*
*הצג את הערך המחושב*

עבור רשימה של 100 מחירים ניתן לכתוב אלגוריתם הכולל הוראה לביצוע-חוזר במבנה זהה, אלא שמספר הפעמים המצוין בו בכותרת ההוראה הוא 100 במקום 10.
**שימו ♥** להזחה בהוראה לביצוע-חוזר. בהוראה זו (כמו בהוראה לביצוע-בתנאי) אנו מזיחים פנימה את קבוצת ההוראה לביצוע-חוזר.

## בחירת משתנים

נשתמש במשתנים הבאים מטיפוס ממשי:
**rate** – ישמור את שער ההמרה מדולרים לשקלים.
**dollarPrice** – ישמור מחיר בדולרים
**shekelPrice** – ישמור את ערכו בשקלים של המחיר השמור ב-dollarPrice

## האלגוריתם

1. *קלוט שער המרה ב-*rate
2. *בצע 10 פעמים:*
   2.1. *קלוט מחיר ב-*dollarPrice
   2.2. *חשב את ערכו בשקלים של המחיר השמור ב-*dollarPrice *והשם ב-*shekelPrice
   2.3. *הצג את ערכו של* shekelPrice

## יישום האלגוריתם

הוראה לביצוע-חוזר במבנה של ***בדיוק מספר פעמים...*** מיושמת ב-Java במשפט **for**. משפט **for** משתמש **במשתנה בקרה**, אשר שולט בביצוע הלולאה. למשל אם ברצוננו לבצע קבוצת הוראות 10 פעמים, נכתוב משפט **for** בצורה הבאה:

```
for (i = 1; i <= 10; i++)
{
    ההוראות לביצוע
}
```

משתנה הבקרה במשפט זה הוא i. ערכו מאותחל ב-1 (כפי שמורה משפט ההשמה `i = 1`, המהווה את הרכיב הראשון בסוגריים). אחרי שסדרת ההוראות לביצוע מתבצעת פעם אחת ערכו של משתנה הבקרה גדל ב-1 (על כך מורה הרכיב השלישי בסוגריים: `i++`), והביצוע-החוזר יימשך כל עוד ערכו של i קטן או שווה ל-10 (כפי שמורה התנאי `i <= 10`, הרכיב השני בסוגריים).

אם כך בתחילת הביצוע של משפט ה-**for**, i יאותחל ב-1. אחרי שקבוצת ההוראות לביצוע תבוצע פעם אחת, ערכו יגדל ל-2. אחרי שקבוצת ההוראות תבוצע פעם שנייה, ערכו יגדל ל-3. אחרי שקבוצת ההוראות תבוצע פעם עשירית, ערכו כבר יהיה 11, ואז יסתיים הביצוע-החוזר, משום שערכו של התנאי `i <= 10` יהיה false.

באופן כללי, בביצוע משפט **for** מושם ערך התחלתי במשתנה הבקרה לפי הרכיב הראשון במשפט. לאחר מכן מתבצעת בדיקת התנאי, המתואר ברכיב השני. אם ערכו של התנאי הוא true מתבצעות ההוראות לביצוע. בתום ביצוע קבוצת ההוראות גדל ערכו של משתנה הבקרה לפי הרכיב השלישי. כעת מתבצעת שוב בדיקת התנאי. אם ערכו של התנאי הוא true מתבצעת שוב קבוצת ההוראות וכך הלאה, עד אשר ערכו של התנאי הוא false ואז מסתיים הביצוע.

באופן דומה, ניתן היה לבחור גם במשפט ה-**for** הבא ליישום ההוראה לביצוע-חוזר באלגוריתם:

```
for (i = 0; i < 10; i++)
{
    הוראות לביצוע
}
```

גם במקרה זה ההוראות לביצוע מתבצעות 10 פעמים: פעם אחת כאשר ערכו של i שווה ל-0, פעם שנייה כאשר ערכו שווה ל-1, פעם שלישית כאשר ערכו שווה ל-2, ובפעם העשירית ואחרונה כאשר ערכו של i שווה ל-9. כאשר ערכו של i גדל שוב, ומגיע ל-10, התנאי להמשך הביצוע כבר לא מתקיים, והביצוע-החוזר מסתיים.

**שימו ♥:** ההוראה `i++` היא למעשה הוראת השמה מקוצרת. היא שקולה להוראת ההשמה: `i = i + 1`. ניתן להשתמש בהוראה זו בכל מקום בתוכנית, לאו דווקא במשפט **for**.

## התוכנית המלאה

```
/*
הקלט: שער ההמרה מדולרים לשקלים ורשימה של 10 מחירים בדולרים
הפלט: הערכים השקליים של 10 המחירים הנתונים בדולרים
*/
import java.util.Scanner;
public class Convertor
{
    public static void main (String [] args)
    {
        final int HOW_MANY=10; //קבוע: מספר המחירים הנקראים מהקלט
```

```
        double rate;            //שער ההמרה
        double dollarPrice;     //מחיר בדולרים
        double shekelPrice;     //מחיר בשקלים
        int i;                  //משתנה בקרה
        Scanner in = new Scanner(System.in);
        // קלט
1.      System.out.print("Enter the rate: ");
2.      rate = in.nextDouble();
        //   ההוראה לביצוע-חוזר
3.      for (i = 1; i <= HOW_MANY; i++)
        {
3.1         System.out.print("Enter price in Dollars: ");
3.2        dollarPrice = in.nextDouble();
3.3        shekelPrice = dollarPrice * rate;
3.4        System.out.println("Price in Shekels is " + shekelPrice);
        }// for
    }// main
}// Convertor
```

## מעקב

בטבלת מעקב הכוללת משפט **for** נכלול עמודה עבור משתנה הבקרה של המשפט. בתוכנית לפתרון בעיה 1, ערכו של משתנה הבקרה i גדל ב-1 אחרי כל ביצוע של קבוצת ההוראות הכלולה במשפט ה-**for**. ערכו בביצוע-החוזר הראשון הוא 1 וערכו בביצוע-החוזר האחרון הוא 10. בנוסף לכך, טבלת המעקב תכלול עמודה עבור התנאי הבוליאני שבכותרת המשפט.

נעקוב אחר ביצוע התוכנית עבור הקלט הבא: שער ההמרה הוא 3, ועשרת המחירים להמרה הם: 100 61.03 71.05 60 50.3 200.01 120.3 18.2 5 10.1.

כדי להימנע מהצגת טבלה ארוכה מדי, תכלול הטבלה הבאה מעקב רק אחרי עיבוד שני המחירים הראשונים והמחיר האחרון שבקלט.

| | **המשפט לביצוע** | i | i<=10 | rate | dollar | shekel | **פלט** |
|---|---|---|---|---|---|---|---|
| 1 | System.out.print("Enter the rate: "); | ? | | ? | ? | ? | Enter the rate |
| 2 | rate=in.nextdouble(); | ? | | 3 | ? | ? | |
| 3 | **for** (i = 1; i <= 10; i++) | 1 | true | 3 | ? | ? | |
| 3.1 | System.out.print("Enter price in dollars: "); | 1 | | 3 | ? | ? | Enter price … |
| 3.2 | dollarPrice = in.nextDouble(); | 1 | | 3 | 10.1 | ? | |
| 3.3 | shekelPrice = dollarPrice * rate; | 1 | | 3 | 10.1 | 30.3 | |
| 3.4 | System.out.println("Price in Shekels is " + shekelPrice); | 1 | | 3 | 10.1 | 30.3 | Price in Shekels is 30.3 |
| 3 | **for** (i = 1; i <= 10; i++) | 2 | true | 3 | 10.1 | 30.3 | |
| 3.1 | System.out.print("Enter | 2 | | 3 | 10.1 | 30.3 | Enter price … |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | price in dollars: "); | | | | | | |
| 3.2 | dollarPrice = in.nextDouble(); | 2 | | 3 | 5 | 30.3 | |
| 3.3 | shekelPrice = dollarPrice * rate; | 2 | | 3 | 5 | 15 | |
| 3.4 | System.out.println("Price in Shekels is " + shekelPrice); | 2 | | 3 | 5 | 15 | Price in Shekels is 15 |
| . . . | | | | | | | |
| 3 | **for** (i = 1; i <= 10; i++) | 10 | true | 3 | 61.03 | 183.09 | |
| 3.1 | System.out.print("Enter price in dollars: "); | 10 | | 3 | 61.03 | 183.09 | Enter price … |
| 3.2 | dollarPrice = in.nextDouble(); | 10 | | 3 | 100 | 183.09 | |
| 3.3 | shekelPrice = dollarPrice * rate; | 10 | | 3 | 100 | 300 | |
| 3.4 | System.out.println("Price in Shekels is " + shekelPrice); | 10 | | 3 | 100 | 300 | Price in Shekels is 300 |
| 3 | **for** (i = 1; i <= 10; i++) | 11 | false | 3 | 100 | 300 | |

שימו לב לשינויים שחלים בערכו של משתנה הבקרה i ולבדיקת התנאי של משתנה הבקרה.

### סוף פתרון בעיה 1

נציג את המושגים החדשים שהכרנו בפתרון לבעיה 1.

באלגוריתם לפתרון הבעיה כללנו **הוראה לביצוע-חוזר במבנה בצע מספר פעמים**... כדי לציין ביצוע-חוזר של תת-משימה. הוראה זו מורה על ביצוע-חוזר של קבוצת הוראות מספר פעמים. הוראה לביצוע-חוזר היא הוראת בקרה.

הוראה לביצוע-חוזר נקראת גם **לולאה** (loop), וקבוצת ההוראות לביצוע הכלולות בה נקראת **גוף הלולאה**.

בדומה לכתיבה של הוראה לביצוע-בתנאי גם כתיבה של הוראה לביצוע-חוזר נעשית תוך הקפדה על **הזחה** מתאימה: קבוצת ההוראות שיש לחזור על ביצוען מוזחת פנימה.

בשפת Java מיושמת הוראה לביצוע-חוזר במבנה זה במשפט **for**.

זהו **המבנה הכללי של משפט for** בשפת Java:

```
for (שינוי משתנה הבקרה ;התנאי להמשך הביצוע ;אתחול משתנה הבקרה)
{
    הוראות לביצוע
}
```

**אתחול משתנה הבקרה**: הוראת השמה הקובעת ערך התחלתי למשתנה הבקרה.

**התנאי להמשך הביצוע**: ביטוי בוליאני שמהווה את התנאי השולט בביצוע-החוזר. התנאי נבדק אחרי אתחול משתנה הבקרה. כמו כן הוא נבדק שוב בכל פעם שמסתיים ביצוע של קבוצת ההוראות-לביצוע. כל עוד התנאי מתקיים הביצוע-החוזר ממשיך. כאשר ערכו של התנאי הוא false הביצוע-החוזר מסתיים.

**שינוי משתנה הבקרה**: השינוי שחל במשתנה הבקרה בכל פעם שמסתיים שלב ביצוע נוסף.

**גוף הלולאה** תחום בסוגריים מסולסלים. במקרה שגוף הלולאה מכיל משפט בודד אפשר להשמיט את הסוגריים.

למשל כך:

```
for (i = 0; i < 10; i++)
    System.out.print("*");
```

ב**טבלת מעקב** אחר תוכנית הכוללת משפט **for** אנו כוללים עמודה עבור משתנה הבקרה ועמודה עבור התנאי להמשך הביצוע.

משתנה הבקרה במשפט **for** הוא בדרך כלל מטיפוס שלם. מאחר שבמקרים רבים תפקידו של משתנה הבקרה הוא רק לשלוט במשפט ה-**for**, ואין בו שימוש בחלקי התוכנית האחרים, שפת Java מאפשרת להצהיר על משתנה הבקרה בתוך הוראת ה- **for**, כלומר, נוכל לכתוב:

```
for (int i = 1; i <= 10; i++)
```

כאשר מצהירים על משתנה הבקרה בתוך משפט ה-**for**, אין אליו גישה מחוץ לתחום משפט ה-**for**.

---

**שאלה 7.1**

בנו טבלת מעקב אחר מהלך ביצוע התוכנית `Convertor` (לפתרון בעיה 1) עבור הקלט שבו שער ההמרה הוא 4.5 ועשרת המחירים להמרה הם:

5.05 18.01 17.03 20.9 101 105 213.05 16.1 17.2 18.3

פרטו בטבלה רק את השורות המתאימות לעיבוד שני הקלטים הראשונים ולעיבוד הקלט האחרון (בדומה לנעשה בפתרון בעיה 1).

**שאלה 7.2**

נסחו עבור כל אחת מן הבעיות האלגוריתמיות הבאות קבוצת תת-משימות לביצוע-חוזר:

א. הקלט הוא 20 מרחקים הנתונים במיילים, והפלט הוא 20 המרחקים בקילומטרים (1 מייל = 1.6 קילומטר).

ב. הקלט הוא עשר אותיות מן הא״ב האנגלי השונות מהאות $Z$, והפלט הוא עשר האותיות שעוקבות לאותיות הנתונות.

ג. הקלט הוא 40 זוגות של ציונים (זוג ציונים עבור כל תלמיד), והפלט הוא רשימה של ארבעים מספרים: כל מספר הוא הממוצע של זוג הציונים המתאים לו.

**שאלה 7.3**

לפניכם קטע תוכנית:

```
System.out.print('X');
for (i = 0; i < 10; i++)
    System.out.print("*");
System.out.print('X');
```

מהו פלט קטע התוכנית?

**שאלה 7.4**

פתחו וישמו אלגוריתם אשר הקלט שלו הוא תו, והפלט שלו הוא שכפול של התו הנקלט חמישים פעמים. למשל, עבור הקלט A יהיה הפלט AAA...AA (חמישים פעמים).
בשלב החלוקה לתת-משימות הקפידו על ניסוח תת-משימה לביצוע-חוזר.

**שאלה 7.5**

פתחו וישמו אלגוריתם אשר הקלט שלו הוא 20 מספרים שלמים חיוביים דו-ספרתיים, והפלט שלו הוא סכום הספרות לכל אחד מהמספרים הנתונים.
למשל, אם הקלט הוא:

11 17 99 10 20 30 10 20 30 10 20 30 10 10 20 20 30 30 88 15

הפלט המתאים הוא:

2 8 18 1 2 3 1 2 3 1 2 3 1 1 2 2 3 3 16 6

בשלב החלוקה לתת-משימות הקפידו על ניסוח תת-משימה לביצוע-חוזר, ובשלב הבדיקה הקפידו על בדיקה מסודרת באמצעות טבלת מעקב.

---

בפתרון בעיה 1 מספר הפעמים לביצוע-חוזר נקבע עוד לפני תחילת ביצוע התוכנית. במקרים רבים, ייתכן כי מספר הפעמים לביצוע-חוזר ידוע לפני שמתחיל ביצועה של ההוראה לביצוע-חוזר, אך לא לפני תחילת ביצוע התוכנית. כלומר הוא תלוי בקלט. מקרה כזה מודגם בבעיה הבאה:

## בעיה 2

**מטרת הבעיה ופתרונה:** הצגת ביצוע-חוזר שאורכו נקבע על פי נתון קלט.

| |
|---|
| פתחו אלגוריתם שיקבל כקלט מספר שלם N, ויציג על המסך שורה של כוכביות באורך N. |

### פירוק הבעיה לתת-משימות

1. קליטת אורך לרשימת כוכביות
2. הדפסת כוכביות לפי המספר הנקלט

### רשימת המשתנים

עד עתה השתמשנו במספרים קבועים בתנאי לסיום הביצוע-החוזר. בבעיה זו אנו נדרשים לקבל כקלט את מספר הפעמים שתתבצע הלולאה, ולכן התנאי להמשך הביצוע יהיה תלוי בערכו של משתנה.

**numOfTimes** – מספר הפעמים שיש להציג כוכבית
**i** – משתנה הבקרה של הלולאה

### האלגוריתם

1. קלוט מספר שלם ב-numOfTimes
2. בצע numOfTimes פעמים:
   2.1. הצג '*'

### יישום האלגוריתם

בפתרון בעיה 1, כאשר רצינו לבצע קבוצת משימות 10 פעמים יישמנו זאת באמצעות משפט **for** שכותרתו:

```
for(i = 1; i <= 10; i++)
```

(או באמצעות משפט שכותרתו **for**(i = 1; i <= HOW_MANY; i++), ו-HOW_MANY הוגדר כקבוע שערכו 10).

כעת אנחנו רוצים לבצע את הוראה numOfTimes 2.1 פעמים. לכן ניישם את ההוראה לביצוע-חוזר במשפט **for** שכותרתו:

```
for(i = 1; i <= numOfTimes; i++)
```

### התוכנית המלאה

```
/*
הקלט: מספר שלם
הפלט: שורה של כוכביות באורך הנתון כקלט
*/
import java.util.Scanner;
public class Stars
{
    public static void main (String [] args)
    {
            int numOfTimes;
            Scanner in = new Scanner(System.in);
            System.out.print("Enter a number please");
            numOfTimes = in.nextInt();
            for (int i = 1; i <= numOfTimes; i++)
            System.out.print('*');
    } // main
} // Stars
```

***סוף פתרון בעיה 2***

בתוכנית Stars ראינו שימוש בנתון קלט לקביעת מספר הפעמים לביצוע לולאה. בפיתוח אלגוריתמים בהמשך נשתמש בכך פעמים רבות.

באמצעות הבעיה הבאה נכיר שני סוגים של משתנים המשמשים בפתרון בעיות רבות ותבנית העבודה איתם היא שימושית מאוד. כפי שנראה, תבנית העבודה עם משתנים כאלה משתמשת בהוראות לביצוע-חוזר.

## *בעיה 3*

**מטרת הבעיה ופתרונה**: הצגת **מונה** ו**צובר** ואופן העבודה איתם.

פתחו וישמו אלגוריתם אשר הקלט שלו הוא מספר חיובי שלם, ולאחריו רשימה של מספרים ממשיים, שאורכה שווה לערך הקלט הראשון. הפלט שלו הוא ממוצע המספרים ברשימה ומספר המספרים ברשימה שערכם עולה על 50.
למשל, עבור הקלט:

```
10 23.4 100 95 78 64.15 75 90.3 54.2 67 20
```

הפלט המתאים הוא: 8 66.7

משום שממוצע המספרים הוא 66.7, ו-8 מתוכם הם גדולים מ-50.

## פירוק הבעיה לתת-משימות

1. קליטת אורך הרשימה
2. קליטת רשימת הערכים, סיכומם, ומניית מספר הערכים הגבוהים מ-50
3. חישוב ממוצע הערכים
4. הצגה כפלט של הממוצע שחושב ושל מספר הערכים הגבוהים מ-50

**?** כדי לחשב את הממוצע יש לסכם את הערכים הנתונים בקלט, ובנוסף יש למנות כמה מהערכים הנתונים בקלט הם גבוהים מ-50. ניתן לבצע הן את פעולת הסיכום והן את פעולת המנייה תוך כדי קריאת נתוני הקלט. כיצד ניתן לבטא זאת כהוראה לביצוע-חוזר?

ניתן לבצע את פעולת **הסיכום** בהוספת כל ערך שנקלט לסכום מצטבר, עוד לפני קריאת הערך הבא. בדומה, ניתן לבצע את פעולת **המנייה** בהשוואת כל ערך שנקלט ל-50, ובהגדלת מונה מתאים בכל פעם שהערך שנקלט עולה על 50. לכן, את תת-משימה 2 נוכל לבצע באמצעות ביצוע-חוזר של קבוצת ההוראות הבאה:

2.1. קליטת ערך
2.2. הוספת הערך שנקלט לסכום המצטבר
2.3. השוואת הערך שנקלט ל-50. אם גבוה מ-50, הגדלת ערכו של מונה למניית מספר הערכים הגבוהים מ-50.

## בחירת משתנים

מן התת-משימות לביצוע-חוזר ניתן להסיק שיש להשתמש במשתנה אשר יישמר בו ערך תורן שנקרא מהקלט, במשתנה שיצבור את **סכום** הערכים, ובמשתנה ש**ימנה** את מספר הערכים הגבוהים מ-50. לכן ניעזר במשתנים הבאים:

**length** – מטיפוס שלם, ישמור את אורך רשימת המספרים
**num** – מטיפוס ממשי, ישמור ערך תורן הנקרא מהקלט
**sum** – מטיפוס ממשי, **צובר** שישמור את סכום הערכים
**average** – מטיפוס ממשי, ישמור את ממוצע הערכים
**counterLarge** – מטיפוס שלם, **מונה** שישמור את מספר הערכים הגבוהים מ-50

**שימו ♥:** המשתנים average, num, sum הם מטיפוס ממשי, כיוון שערכי הקלט בבעיה הם ממשיים. לעומתם counterLarge הוא מטיפוס שלם כיוון שהוא משמש למנייה.

## יישום האלגוריתם

את ההוראה לביצוע-חוזר נוכל לנסח באופן הבא:

*בצע* length *פעמים:*
1. *קלוט ערך ממשי ב-*num
2. *הוסף את ערכו של* num *לסכום המצטבר השמור ב-*sum
3. *אם ערכו של* num *גדול מ-*50
   3.1. *הגדל ב-*1 *את ערכו של* counterLarge

**?** אילו הוראות יש להוסיף **לפני** ההוראה לביצוע-חוזר?

**שימו ♥:** כדי שהצבירה והמנייה יתבצעו באופן נכון, עלינו לאתחל נכונה את הערכים של sum ושל counterLarge. על פי תפקידם של שני המשתנים באלגוריתם יש לאתחל את ערכיהם ב-0.

### התוכנית המלאה

```
/*
קלט: רשימת ערכים ממשיים
פלט: ממוצע הערכים ומספר הערכים הגבוהים מ-50
*/
import java.util.Scanner;
public class CalcAvgAndCountLargerThan50
{
    public static void main (String [] args)
    {
        // הצהרה על קבוע בתוכנית
        final int LIMIT = 50;
        // הצהרה על משתנים בתוכנית
        int length;            //אורך רשימת הקלט
        double num;      //ערך קלט תורן
        double sum = 0;  //צובר
        double average;  //ממוצע
        int counterLarge = 0;  //מונה למספר הערכים הגבוהים מ-50
        Scanner in = new Scanner(System.in);
        System.out.print("Enter length of input list: ");
        length = in.nextInt();
        // ההוראה לביצוע-חוזר
        for (int i = 1; i <= length; i++)
        {
               System.out.print("Enter a number: ");
            num = in.nextDouble();
            sum = sum + num;
            if (num > LIMIT)
                counterLarge++;  // counterLarge=counterLarge+1 -שקול ל
        }// for
        average = sum / length;
        System.out.println("Average is " + average);
        System.out.println(counterLarge + " numbers are larger than "
                                                        + LIMIT);
    }// main
}// CalcAvgAndCountLargerThan50
```

### סוף פתרון בעיה 3

באלגוריתם לפתרון בעיה 3 משמש המשתנה sum **כצובר** של הערכים הנתונים.

| |
|---|
| צובר הוא משתנה אשר תפקידו לצבור ערכים. למשל ניתן להשתמש בצובר לסכימת ערכים. |

המשתנה counterLarge משמש **כמונה** של מספר הערכים הגבוהים מ-50 מבין הערכים הנתונים.

| |
|---|
| מונה הינו משתנה אשר תפקידו למנות את מספר הפעמים שהתרחש אירוע מסוים (למשל מספר הפעמים שנקרא נתון קלט). כיוון שהשימוש במונה הוא לספירה, מונה הוא משתנה מ**טיפוס שלם**. |

בתוכנית לפתרון בעיה 3 מופיעים המשפטים המבטאים את פעולות הצבירה והמנייה בגוף הלולאה לאחר משפט הקלט.
המשפט המבטא את **פעולת הצבירה** בגוף הלולאה הוא:

```
sum = sum + num;
```

המשפט המבטא את **פעולת המנייה** בגוף הלולאה הוא:

```
counterLarge++;
```

מאחר שתחזוקה של מונה או של צובר כוללת ביצוע של פעולות עדכון חוזרות ונשנות, נבצע בדרך-כלל פעולות צבירה ומנייה בגוף הלולאה.

**שימו ♥** בשני המקרים גדל המשתנה המשמש כצובר או כמונה בערך כלשהו. במקרה של צובר הוא גדל בערך ששייך לקבוצת הערכים המצטברים. במקרה של מונה הוא גדל ב-1.

באלגוריתם שמשתמשים בו בצובר או במונה יש לאתחל את הצובר או את המונה. בתוכנית לפתרון הבעיה מאותחלים הצובר `sum` והמונה `counterLarge` ב-0.

**אִתְחוּל של צובר או של מונה** הוא השמת ערך המתאים לתחילת תהליך הצבירה או המנייה. מונה מאותחל בדרך כלל ב-0. הערך ההתחלתי המתאים לצובר תלוי במהות הצבירה המתבצעת בו. למשל, צובר השומר סכום מצטבר יאותחל ב-0.

---

**שאלה 7.6**

שנו את המשפטים הנמצאים בגוף הלולאה שבתוכנית לפתרון בעיה 3, כך שיחושב ממוצע הערכים הגבוהים מ-50.

**שאלה 7.7**

לפעמים ניתן להשתמש בערכו של מונה לחישוב מספר הנתונים המאופיינים בצורה הפוכה לנתונים שמנינו. למשל, נניח שבבעיה 3 יש להציג גם את מספר הערכים ה**קטנים או שווים** ל-50.
דרך אחת לחישוב מספר זה היא באמצעות שימוש במונה נוסף `counterSmall` (נוסף ל-`counterLarge`), אשר ערכו יוגדל ב-1 בכל פעם שנקלוט ערך הקטן או שווה ל-50.
אך בעצם אין צורך במונה נוסף. ניתן לבצע את החישוב בתום ביצוע הלולאה, באמצעות המונה `counterLarge` המופיע כבר בתוכנית. כיצד? הוסיפו לתוכנית את ההוראה או את ההוראות המתאימות.

**שאלה 7.8**

ציינו עבור כל אחת מן הבעיות האלגוריתמיות הבאות אם נחוץ לפתרונה צובר, מונה או אף אחד מהשניים:

א. קלט: רשימת מחירים, פלט: סך כל המחירים.
ב. קלט: רשימת מחירים, פלט: מספר המחירים הגבוהים מ-100.
ג. קלט: רשימת מחירים, פלט: מספר המחירים שמרכיב האגורות בהם שונה מ-0.
ד. קלט: רשימת מספרים, פלט: הערך המוחלט של כל אחד מהמספרים.
ה. קלט: רשימת מספרים, פלט: הממוצע של הערכים המוחלטים של המספרים.

**שאלה 7.9**

יש לקלוט סדרה של תווים, אשר אורכה שמור במשתנה `listSize`, ולמנות את מספר התווים שהם אותיות גדולות (capital letters) באי"ב האנגלי.
בחרו משתנים, כתבו הוראה לביצוע-חוזר אשר לפניה אתחול מתאים, וישמו אותה במשפט **`for`**.

**שאלה 7.10**

מטרת קטע התוכנית הבא היא מניית מספר תווי קלט השונים מן האות A. המשתנים `counter` ו-`listSize` הם מטיפוס שלם והמשתנה `letter` הוא מטיפוס תווי.

```
int counter = ___________ ;
for (int i = 1; i <= listSize; i++)
{
      System.out.print("Enter a char: ");
     letter = in.next().charAt(0);
     if (_____________________________)

        _____________________________
}// for
System.out.println("There are " + counter
                                  + "letters different than A");
```

קטע התוכנית כולל לולאה.

א. כמה פעמים תתבצע הלולאה עבור הערך 10 ב-`listSize`?

ב. כמה פעמים תתבצע הלולאה עבור הערך 1 ב-`listSize`?

ג. השלימו את קטע התוכנית.

**שאלה 7.11**

נתונה התוכנית הבאה:

```
/*
קלט: טור של 16 תווים מטופס ספורטוטו
_____________________________________:פלט
*/
import java.util.Scanner;
public class Toto
{
    public static void main (String [] args)
    {
        final int NUM_OF_GAMES = 16;
        int d = 0;
        char score;
        Scanner in = new Scanner(System.in);
        for (int i = 0; i < NUM_OF_GAMES; i++)
        {
              System.out.print("Enter the score: ");
            score = in.next().charAt(0);
            if (score == 'X')
                d++;
        } // for
        System.out.println(d);
    } // main
} // Toto
```

קלט התוכנית הוא טור בטופס הספורטוטו. כלומר, 16 תווים שכל אחד מהם מציין תוצאת משחק (1, 2 או X).

א. מהו הפלט עבור הקלט: 1 2 X 1 2 X 1 2 X 1 2 X 1 2 X 1 2?

ב. האם התוכנית כוללת מונה או צובר? אם כן, מהו או מהם?

ג. כמה פעמים תתבצע לולאת התוכנית?
ד. הביאו דוגמת קלט אשר הפלט עבורה הוא 0.
ה. הביאו דוגמת קלט אשר הפלט עבורה הוא 15.
ו. מה מאפיין את הקלטים אשר הפלט עבורם הוא 1?
ז. מהם ערכי הפלט האפשריים?
ח. מהי מטרת התוכנית? בחרו שם משמעותי למשתנה d.

**שאלה 7.12**

פתחו וישמו אלגוריתם אשר הקלט שלו הוא מספר המציין אורך של רשימה ואחריו רשימת מספרים ממשיים באורך הנתון. הפלט שלו הוא סכום החלקים השלמים של המספרים הממשיים הנתונים.

**שאלה 7.13**

כתבו קטע תוכנית אשר מקבל כקלט מספר שלם num. התוכנית תגריל num מספרים בתחום 10-50 ותבדוק כמה מספרים מבין המספרים שהוגרלו הם זוגיים.

**שאלה 7.14**

כתבו קטע תוכנית שיגריל 10 מספרים תלת-ספרתיים. התוכנית תחשב ותדפיס את סכום המספרים האי-זוגיים בלבד מבין המספרים שהוגרלו.

---

בדוגמאות שראינו עד כה השתמשנו בצובר לצבירת סכום. ניתן להשתמש בצובר גם לצבירת מכפלה. כאשר צובר משמש לצבירת מכפלה הוא מאותחל בערך שונה מ-0. שימו לב לכך בשאלה הבאה:

---

**שאלה 7.15**

פתחו וישמו אלגוריתם שיקבל כקלט רשימה של 50 מספרים ויציג כפלט את **מכפלתם** של המספרים הקטנים מ-10.

---

להעמקה בתבניות ***מניה*** ו***צבירה*** פנו לסעיף התבניות המופיע בסוף הפרק.

בעיה 3 השתמשה גם בתבנית ***ממוצע***. להעמקה בתבנית ***ממוצע*** עבור סדרה שיכולה להכיל יותר מ-3 ערכים פנו לסעיף התבניות המופיע בסוף הפרק.

## *בעיה 4*

**מטרת הבעיה ופתרונה**: עוד על הוראה לביצוע-חוזר מספר פעמים ידוע מראש: הצגת הוראה לביצוע-חוזר המטפלת בתחום של מספרים, והדגמת שימוש במשתנה הבקרה בתוך גוף הלולאה.

פתחו וישמו אלגוריתם שיקבל כקלט מספר שלם, ויציג כפלט את כל המספרים השלמים החיוביים הקטנים מהמספר הנתון. למשל עבור הקלט 5 הפלט המתאים הוא: 4 3 2 1.

### פירוק הבעיה לתת-משימות

ברצוננו לקלוט מספר שלם num ולהציג num-1 ערכים, מ-1 עד num-1. אם כך, החלוקה לתת-משימות היא ברורה למדי:

1. קליטת מספר מהקלט

2. הצגה כפלט של כל הערכים החיוביים והשלמים שקטנים מהמספר שנקלט

ברור כי לשם ביצוע תת-משימה 2 נזדקק להוראה לביצוע-חוזר, שתתבצע num-1 פעמים.

## בחירת משתנים

ברור כי נזדקק למשתנה עבור הערך הנקרא מהקלט. סביר כי נהיה גם זקוקים למשתנה כלשהו, עבור האיבר התורן להצגה. משתנה זה יאותחל ב-1 ויקודם בסיום כל סיבוב בלולאה ב-1, עד אשר ערכו יגיע ל-num.

אבל איננו זקוקים למשתנה **נוסף** כי תיאור זה מתאים בדיוק למשתנה הבקרה של הלולאה!

לכן נקבל את רשימת המשתנים הבאה:

**num** – מטיפוס שלם, לשמירת הערך הנקרא מהקלט

**i** – משתנה הבקרה

## האלגוריתם

כאמור תת-משימה 2 תתבצע באמצעות הוראה לביצוע-חוזר. למעשה בהוראה זו אנו מעוניינים לעבור על תחום של ערכים (מהערך 1 ועד הערך num-1) ולהציג כל ערך בתחום. כדי לבטא זאת, ננסח את ההוראה לביצוע-חוזר בצורה שונה מעט מזו שראינו באלגוריתמים קודמים בפרק.

1. קלוט מספר שלם ל-num
2. עבור כל i שלם מ-1 עד num-1 בצע:
   2.1. הצג את ערכו של i

## יישום האלגוריתם

את הוראה 2 באלגוריתם שכתבנו ניישם במשפט **for**. זהו משפט הדומה מאוד לאלה שראינו בתוכניות קודמות בפרק, רק שמשתנה הבקרה בו אינו משמש רק כדי לשלוט בביצוע הלולאה. יש התייחסות למשתנה הבקרה גם בתוך גוף הלולאה, ולא רק בשלושת הרכיבים שבכותרת הלולאה.

## התוכנית המלאה

```
/*
קלט: מספר שלם
פלט: כל המספרים השלמים והחיוביים הקטנים מהמספר הנתון
*/
import java.util.Scanner;
public class PrintNumbers
{
    public static void main(String[] args)
    {
        int num;        // ערך הנקרא מהקלט
        Scanner in = new Scanner(System.in);
          System.out.print("Enter a number: ");
        num=in.nextInt();
        for (int i = 1; i < num; i++)
            System.out.print(i + " ");
    } // main
} // PrintNumbers
```

*סוף התרון בעזרת 4*

**שאלה 7.16**

על פי הגדרת בעיה 4, יכול להינתן כנתון ראשון בקלט כל מספר שלם. תארו את מהלך ביצוע התוכנית `PrintNumbers` אם הערך הראשון בקלט הוא המספר השלם 0. תארו את מהלך הביצוע של התוכנית אם הערך הראשון בקלט הוא המספר השלם 2-.

**שאלה 7.17**

פתחו אלגוריתם שמקבל כקלט מספר חיובי שלם n, מחשב את !n, ומציג את הערך שחושב כפלט. ישמו את האלגוריתם בשפת Java.

להזכירכם: !n היא מכפלת המספרים מ-1 עד n, כלומר: $n! = 1 \cdot 2 \cdot \ldots \cdot n$.

**שאלה 7.18**

פתחו אלגוריתם שמקבל כקלט מספר חיובי שלם n, ומחשב את הסכום: $\frac{1}{1}+\frac{1}{2}+\ldots+\frac{1}{n}$.

ישמו את האלגוריתם בשפת java.

---

כזכור הרכיב השלישי במשפט **for** הוא משפט השמה המתאר את השינוי של משתנה הבקרה בתום כל סיבוב בלולאה. עד עתה כתבנו תוכניות שבהן משתנה הבקרה גדל ב-1 בתום כל סיבוב של הלולאה. אבל לעתים נרצה לקדם את משתנה הבקרה ב-2, 3 או אפילו פי 2. שפת Java מאפשרת לנו לתאר שינויים כאלה, פשוט בכתיבת משפט השמה מתאים. הנה מספר דוגמאות לכותרות משפטי **for** כאלה:

```
for(int i = small; i < big; i = i + 2)
for(int i = small; i < big; i = i * 2)
for(int i = big; i > small; i--)
```

(כפי שוודאי הבנתם, `i--` היא דרך מקוצרת לכתוב: `i = i - 1`).

במקרים רבים, אכן מתאים לבצע עדכונים שונים של משתנה הבקרה, כפי שמדגימות השאלות הבאות.

---

**שאלה 7.19**

לפניכם קטע תוכנית הכתוב ב-java

```
for (int i = 1; i <= 50; i = i * 2)
    System.out.println(i);
```

א. עקבו בעזרת טבלת מעקב אחר קטע התוכנית? מה יודפס?

ב. נניח כי במקום הערך 50 מופיע ערך חיובי שלם כלשהו N. תארו מה מבצע קטע התוכנית כתלות בערך N.

**שאלה 7.20**

פתחו אלגוריתם אשר מקבל כקלט מספר שלם ומציג כפלט את כל המספרים **הזוגיים** החיוביים הקטנים מהמספר שנקרא.

---

## 7.2 מציאת מקסימום או מינימום

שתיים מהבעיות הבסיסיות ביותר במדעי המחשב הן חישוב הערך הגדול ביותר או הקטן ביותר ברשימה של ערכים נתונים. פתרונן מהווה תבנית שימושית מאוד. בסעיף זה נערוך היכרות עם בעיות אלו ונראה כי כאשר אורך הרשימה ידוע ניתן לפתור אותן בעזרת הוראה לביצוע-חוזר מספר פעמים ידוע מראש.

### בעיה 5

**מטרת הבעיה ופתרונה:** הצגת אופן החישוב של הערך הגדול ביותר ברשימת ערכים נתונים שאורכה נתון אף הוא.

מנהל סניף הבנק החליט לבדוק מהו סכום הכסף הגדול ביותר השמור בחשבונות של לקוחות הסניף. פתחו אלגוריתם אשר קולט את מספר החשבונות בסניף, ולאחר מכן את סכום הכסף הנמצא בכל חשבון וחשבון (כמספר שלם). פלט האלגוריתם יהיה הסכום הגבוה ביותר. ישמו את האלגוריתם בשפת Java.

#### ניתוח הבעיה בעזרת דוגמאות

הרעיון המרכזי בפתרון הבעיה הוא לזכור **בכל זמן** מהלך הביצוע, מהו הערך הגדול ביותר מבין אלה שכבר נקלטו. ערך זה יושווה תמיד לערך הבא שנקרא מהקלט, ויתעדכן בהתאם לתוצאת ההשוואה.

#### פירוק הבעיה לתת-משימות

את הרעיון שתיארנו נוכל לבטא באמצעות התת-משימה הבאה ועל ביצועה יש לחזור לאחר כל קליטת ערך נוסף:

השוואת הסכום האחרון שנקלט לסכום הגדול ביותר שנקלט עד כה, ועדכון הסכום הגדול ביותר שנקרא עד כה על פי תוצאת ההשוואה

#### בחירת משתנים

נזדקק למשתנה שיזכור את מספר ערכי הקלט (מספר החשבונות), ולמשתנה לשמירת הערך התורן בקלט. בנוסף נזדקק למשתנה כדי לזכור את הערך הגדול ביותר מבין אלה שנקלטו. נבחר את המשתנים הבאים מטיפוס שלם:

**howMany** – מספר הערכים ברשימת הנתונים, מספר החשבונות בסניף

**balance** – הערך התורן ברשימת ערכי הקלט

**max** – לשמירת הסכום הגדול ביותר מבין אלה שנקלטו

#### האלגוריתם

max יאותחל בסכום הראשון ברשימה, כיוון שמיד אחרי שנקרא הסכום הראשון, הוא בוודאי הגדול ביותר מבין כל הסכומים שנקראו. כעת יש לקרוא את כל הסכומים האחרים בעזרת לולאת `for`. לאחר כל קליטה של סכום נוסף, יש להשוות את הסכום החדש שנקלט לסכום הגדול ביותר שנקרא עד כה (השמור ב-max). אם הסכום החדש גדול מזה השמור ב-max, יישמר ב-max הסכום החדש.

❓ כמה פעמים צריך להתבצע גוף הלולאה?
הסכום הראשון נקרא מהקלט עוד לפני הלולאה לצורך האתחול של המשתנה max. לכן הלולאה צריכה להתבצע רק howMany-1 פעמים כדי לקלוט ולעבד את ערכי הקלט הנותרים.

## יישום האלגוריתם

```
/*
קלט: מספר חשבונות בנק, ורשימת הסכומים בחשבונות אלה
פלט: הסכום הגבוה ביותר ברשימה
*/
import java.util.Scanner;
public class FindMax
{
    public static void main (String [] args)
    {
        int howMany;     // מספר הערכים ברשימה
        int balance;     // הסכום התורן
        int max;         //הסכום הגבוה ביותר מבין אלה שנקלטו
        Scanner in = new Scanner(System.in);
 1.     System.out.print("Enter the amount of accounts: ");
 2.     howMany = in.nextInt();
 3.     System.out.print("Enter the first balance: ");
 4.     balance = in.nextInt();
 5.     max = balance;         //אתחול המקסימום לסכום הראשון
 6.     for(int i = 2; i <= howMany; i++)
        {
 6.1.       System.out.print("Enter balance of account" + i + ": ");
 6.2.       balance = in.nextInt();
 6.3.       if (balance > max) //הסכום הנוכחי גדול מהמקסימום הנוכחי
 6.3.1.         max = balance;
        } // for
 7.     System.out.println("The maximum is " + max);
    }// main
}// FindMax
```

**שימו ♥** בתוך הלולאה אפשר להשתמש במשתנה הבקרה i כמשתנה לכל דבר.

### סוף פתרון בעיה 5

---

**שאלה 7.21**

בנו טבלת מעקב אחר מהלך ביצוע התוכנית FindMax לפתרון בעיה 5 עבור הקלט:

5 3 2 4 6 -9

כמה פעמים במהלך ביצוע התוכנית מושם ערך במשתנה max?

**שאלה 7.22**

בפתרון בעיה 5 אותחל max לערכו של הנתון הראשון ברשימה. האם הפתרון היה נכון לו max היה מאותחל בערך קבוע כלשהו, למשל 0? הערה: זכרו כי ייתכן שזהו בנק לא מוצלח במיוחד וכל חשבונות הבנק בו במשיכת יתר, כלומר בעלי ערך שלילי.

**שאלה 7.23**

נתון קטע התוכנית החלקי הבא לחישוב המספר **הקטן** ביותר ברשימת מספרים **חיוביים** נתונה, אשר אורכה שמור במשתנה len.

```
min = _____________
System.out.print("Enter the list size: ");
len = in.nextInt();
for (int i = 1; i <= len; i++)
{
      _____________________
      _____________________
      _____________________
}
System.out.println("The minimum is: " + min);
```

השלימו את קטע התוכנית (הוסיפו משתנה או משתנים במידת הצורך).
במשפט הראשון אתחלו את `min` לערך קבוע כלשהו (ולא לערך הראשון ברשימה).

**שאלה 7.24**

פתחו וישמו אלגוריתם שמקבל כקלט רשימה של ציונים במדעי המחשב של 40 תלמידים. הפלט הוא הציון הגבוה ביותר מבין התלמידים **שנכשלו** במבחן (ציון נכשל הוא ציון הנמוך מ-55).
שימו לב: כאן הציון התורן הנקרא מהקלט אינו מושווה בכל מקרה למקסימום הנוכחי, אלא רק כאשר הוא קטן מ-55.

---

כפי שראינו בתרגילים הקודמים, יש דרכים שונות לאתחול בתבנית מציאת מקסימום (וכמובן, כך גם לגבי תבנית מציאת מינימום). המשתנה ששומר את המקסימום התורן, צריך בכל שלב לשמור את האיבר הגדול ביותר מבין האיברים שהושוו עד כה. בדרך כלל נעדיף לאתחל את המשתנה הזה בערך האיבר הראשון ברשימה. אכן אחרי שנקלט איבר אחד בלבד, ודאי שהוא הגדול מבין כל אלה שנקלטו. אך ישנם מקרים שעובדה זו אינה מספיקה ולא נוכל להשתמש באתחול כזה. כך למשל בשאלה 7.24: בשאלה זו אנו מעוניינים למצוא מקסימום רק מבין הציונים הנכשלים ולא מבין כל הציונים שבקלט. כלומר לא כל איבר שנקלט צריך להיות מושווה לצורך מציאת מקסימום. בפרט לא בטוח שהאיבר הראשון הוא ציון נכשל ולכן בכלל לא יהיה מועמד למקסימום. לכן יהיה שגוי לקבוע אותו כמקסימום התחלתי.

במקרים כאלה נוכל להשתמש בקצוות של טווח הערכים האפשריים. למשל אם מדובר בציונים, ערכם יכול לנוע מ-0 עד 100. במקרה כזה, יהיה נכון לאתחל את המקסימום התורן ב-0 (ואת המינימום התורן ב-100). כיוון שברור שהערך הראשון שנשווה למקסימום התורן ערכו הוא 0 לפחות, הרי מיד אחרי ההשוואה הראשונה המקסימום התורן יכיל את האיבר הראשון שהושווה.

**שימו ♥:** לא תמיד ידועים ערכי קצה לטווח הערכים האפשריים! (ראו את שאלה 7.22)

להעמקה בתבניות ***מציאת מקסימום*** ו***מציאת מינימום*** פנו לסעיף התבניות המופיע בסוף הפרק

# 7.3 מציאת ערך נלווה למקסימום או למינימום

בסעיף זה נכיר בעיות הדומות לבעיית מציאת מקסימום או מינימום, אך מעט יותר מורכבות. גם פתרונותיהן מהווים תבניות שימושיות.

## בעיה 6

**מטרת הבעיה ופתרונה:** הצגת אופן חישוב **מיקומו** של הערך הגדול ביותר והקטן ביותר ברשימת ערכים נתונים.

בנגן MP3 שמורים 100 שירים. לכל שיר מספר ייחודי משלו בין 1 ל-100. על צג המכשיר מוצג **מספרו** של השיר הארוך ביותר ושל השיר הקצר ביותר שנמצאים בו. עליכם לפתח אלגוריתם שיאפשר את הצגת המידע הזה. כלומר האלגוריתם יקבל כקלט את רשימת אורכי 100 השירים במכשיר, ופלט האלגוריתם יהיה **מספרו** של השיר הארוך ביותר **ומספרו** של השיר הקצר ביותר.

כמובן שלצורך פתרון הבעיה יש למצוא את השיר הארוך ביותר ואת השיר הקצר ביותר. כלומר יש למצוא ברשימת אורכי השירים ערך מקסימלי וערך מינימלי. אבל בבעיה זו עלינו לשמור מלבד הערך המקסימלי ומלבד הערך המינימלי גם את **מיקומם** של ערכים אלה ברשימה.

## פירוק הבעיה לתת-משימות

את הרעיון המרכזי בפתרון נוכל לבטא באמצעות התת-משימה הבאה אשר על ביצועה יש לחזור לאחר כל קליטת ערך נוסף:

השוואת אורך השיר האחרון שנקלט לאורך השיר הגדול ביותר שנקלט עד כה ולאורך השיר הקצר ביותר שנקלט עד כה, ובהתאם לתוצאת ההשוואה עדכון ערך המקסימום ומיקום המקסימום ועדכון ערך המינימום ומיקום המינימום.

## בחירת משתנים

**currentSongLength** – מטיפוס ממשי, ישמור את אורך השיר הנוכחי.
**longest** – מטיפוס ממשי, ישמור את אורך השיר הארוך ביותר שנקלט עד כה.
**shortest** – מטיפוס ממשי, ישמור את אורך השיר הקצר ביותר שנקלט עד כה.
**placeLongest** – מטיפוס שלם, ישמור את **מיקומו** של השיר הארוך ביותר שנקלט עד כה.
**placeShortest** – מטיפוס שלם, ישמור את **מיקומו** של השיר הקצר ביותר שנקלט עד כה.

**שימו ♥** לבחירת טיפוסי המשתנים: המשתנים longest ו-shortest אשר שומרים את האורך המקסימלי ואת האורך המינימלי חייבים להיות מאותו טיפוס כמו הערכים ברשימת הקלט. כיוון שכאן מדובר באורכי שירים (שיכולים להיות לא שלמים), בחרנו בטיפוס ממשי. לעומתם המשתנים placeLongest ו-placeShortest שומרים מיקום ברשימה, ולכן הם מטיפוס שלם.

## התוכנית המלאה

**?** כיצד נדע מהו מיקומו של שיר נתון?
לשם כך נוכל להשתמש במשתנה הבקרה של משפט ה-**for**. כדי שהלולאה תתבצע 99 פעמים משתנה הבקרה יתקדם מ-2 עד 100, וכך ערכו של משתנה הבקרה יהיה בדיוק מספרו של השיר התורן.

**שימו ♥**: בדיוק כפי שנאתחל את longest ואת shortest, נאתחל גם את placeLongest ואת placeShortest באופן עקבי. Longest יאותחל באורכו של השיר הראשון ולכן בהתאם placeLongest יאותחל במיקום הראשון (כלומר ב-1). כך גם לגבי האתחול של shortest ושל placeShortest.

```
/*
קלט: רשימת אורכי 100 השירים במכשיר MP3
פלט: מספרו של השיר הארוך היותר, ומספרו של השיר הקצר ביותר
*/
```

```
import java.util.Scanner;
public class Mp3
{
    public static void main (String [] args)
    {
        final int NUM_OF_SONGS = 100;// הגדרת קבוע – מספר השירים בנגן
        double currentSongLength; //אורך השיר הנוכחי
        double shortest; // אורך השיר הקצר ביותר
        double longest; //אורך השיר הארוך ביותר
        int placeLongest, placeShortest; // מיקומם של השיר הקצר והארוך
        Scanner in = new Scanner(System.in);
        System.out.print("Enter the length of the first song: ");
        currentSongLength = in.nextDouble();
        // אתחולם של המקסימום, המינימום ומקומם
        longest = currentSongLength;
        shortest = currentSongLength;
        placeLongest = 1;
        placeShortest = 1;
        // הלולאה מתחילה מ-2 כיוון שהשיר הראשון כבר נקלט
        for(int i = 2; i <= NUM_OF_SONGS; i++)
        {
            System.out.print("Enter the length of song " + i + ": ");
            currentSongLength = in.nextDouble();
            if (currentSongLength > longest)
            {
                longest = currentSongLength;
                placeLongest = i;
            } // if
            if (currentSongLength < shortest)
            {
                shortest = currentSongLength;
                placeShortest = i;
            } // if
        } // for
        System.out.println("The number of the longest song is " +
                                                    placeLongest);
        System.out.println("The number of the shortest song is " +
                                                    placeShortest);
    } // main
}// Mp3
```

***סוף פתרון תרגיל 6***

---

**שאלה 7.25**

פתחו אלגוריתם שיקבל כקלט מספר חיובי שלם המציין את מספר שחקני מכבי ת״א, ואחר כך יקלוט רשימה של נתוני קליעות של השחקנים: עבור כל שחקן ייקלט מספרו (המספר שעל החולצה שלו), ומספר הנקודות שקלע במהלך העונה. פלט האלגוריתם יהיה **מספרו** של השחקן שקלע הכי הרבה נקודות במהלך העונה.

**שימו ♥:** במקרה זה, אנחנו לא מתבקשים להציג את מיקומו של השחקן שקלע הכי הרבה נקודות אלא את מספרו, לכן משתנה הבקרה המצביע על ה״מיקום״ של השחקן התורן בקלט אינו מתאים כי הוא אינו מספרו הסידורי של השחקן. את מספרו של השחקן יש לקרוא מהקלט.

---

להעמקה בתבניות ***מציאת ערך נלווה למקסימום*** ו***מציאת ערך נלווה למינימום*** פנו לסעיף התבניות המופיע בסוף הפרק.

## 7.4 ביצוע-חוזר-בתנאי

לצורך פתרון הבעיות שראינו עד כה אפשר היה להשתמש בהוראה לביצוע-חוזר אשר מספר הסיבובים בה נקבע **לפני** תחילת ביצוע הלולאה.

בסעיף זה נכיר בעיות אשר בפתרונן אי אפשר לקבוע לפני תחילת הלולאה את מספר הפעמים לביצוע-חוזר. בפתרון בעיות אלה מהלך הלולאה נקבע בהתאם לתנאי. הלולאה מתבצעת שוב ושוב כל עוד התנאי מתקיים. הלולאה מסתיימת כאשר התנאי לא מתקיים.

### ביצוע-חוזר בשימוש בזקיף

בסעיף זה נראה בעיות דומות לאלו שראינו בסעיפים הקודמים. הדמיון מתבטא בכך שגם בבעיות אלו מתבצע עיבוד של רשימת נתוני קלט. ההבדל נעוץ באורך הרשימה המעובדת. בבעיות שהוצגו עד כה אורך הרשימה היה קבוע של התוכנית או נקרא מהקלט. בבעיות שיוצגו בסעיף זה אורך הרשימה אינו ידוע כלל. במקום זאת האיבר האחרון בקלט הוא איבר מיוחד המסמן את סופה של רשימת ערכי הקלט.

#### בעיה 7

**מטרת הבעיה ופתרונה:** הצגת הוראה לביצוע-חוזר-בתנאי בשימוש בזקיף.

צבי עובד בשעות אחר הצהריים בחלוקת משלוחי פרחים. הוא מעוניין להיעזר במחשב הנייד שהוא נושא עמו כדי לחשב בסיום יום העבודה את סכום הטיפים שהרוויח במהלך היום. בכל פעם שצבי מקבל טיפ הוא מקיש את הסכום שקיבל (בשקלים שלמים). בסיום יום העבודה הוא מקיש 1-. פתחו אלגוריתם המקבל כקלט את רשימת הטיפים שקיבל צבי, המסתיימת בערך 1-, ומחשב את סכום הטיפים הכולל. ישמו את האלגוריתם בשפת Java.

#### ניתוח הבעיה בעזרת דוגמאות

---

**שאלה 7.26**

תארו את הפלט עבור כל אחד מן הקלטים הבאים (משמאל לימין):

א. 1- 9 7 10 12

ב. 1- 20

---

#### פירוק הבעיה לתת-משימות

בניתוח ראשוני של הבעיה ניתן לראות שיש להשתמש בביצוע-חוזר של שתי התת-משימות:

1. קליטת מספר
2. הוספת המספר לסכום (בשימוש בצובר)

לו הקלט היה כולל בתחילתו את מספר הערכים בסדרה הנתונה, היינו מפתחים אלגוריתם דומה לאלגוריתמים שבסעיפים הקודמים, בשימוש בהוראה במבנה ***בצע מספר פעמים***. אבל

מספר הערכים בסדרה לא נתון בתחילת הקלט (כלומר אין אנו יודעים מראש כמה שליחויות עשה צבי באותו היום). במקום זה מופיע בסוף הקלט הערך 1- המציין "סוף קלט". כיצד נשתמש בסימון זה בהוראה לביצוע-חוזר?

נשתמש בערך 1- כדי להחליט על סיום ביצוע-חוזר של התת-משימות שתיארנו. כלומר לאחר קליטה של טיפ תורן, יושווה ערך הטיפ לערך 1-. אם הוא שונה מ-1- הרי הוא נתון קלט רגיל ולכן יש להוסיפו לסכום המצטבר. אחרת יש לסיים את הביצוע-החוזר.

## בחירת משתנים

**tip** – מטיפוס שלם, ישמור את הטיפ למשלוח הנוכחי.
**sum** – מטיפוס שלם, ישמור את סכום הטיפים.

## האלגוריתם

ננסח את הרעיון שתיארנו:

*כל עוד הערך האחרון שנקלט אינו 1- בצע:*
*הוסף את הערך האחרון שנקלט לסכום המצטבר*
*הצג את ערכו של הסכום המצטבר*

בהוראה לביצוע-חוזר מסוג זה נבדק תנאי לפני כל סיבוב בלולאה. במקרה זה התנאי הוא *הערך האחרון שנקלט אינו 1-*. אם התנאי מתקיים מתבצע סיבוב נוסף בלולאה. כאשר התנאי לא מתקיים, כלומר הערך שנקלט הוא אכן 1-, מסתיים הביצוע-החוזר.

**?** בין ההוראות האלו לא נמצאת עדיין הוראה לקליטת הטיפ התורן. היכן נמקם את הוראת הקלט?
יש לבצע את קליטת הטיפ התורן לפני השוואתו לסימן 1-. לכן את הערך הראשון יש לקרוא מהקלט עוד לפני הלולאה. קליטת הערכים הבאים (עד לקליטת סימן סוף הקלט) תהיה ההוראה האחרונה בגוף הלולאה. כלומר מיד כאשר מסתיים עיבוד איבר קלט תורן, ולפני שנבדק קיום התנאי ביחס לאיבר הקלט החדש, מתבצעת קליטה של איבר הקלט החדש.

הנה האלגוריתם המלא:

1. *אתחל את המשתנה* sum *ב-*0
2. *קלוט מספר שלם ל-*tip
3. *כל עוד* tip ≠ 1- *בצע:*
   3.1. *הוסף ל-*sum *את הערך* tip
   3.2. *קלוט מספר שלם ל-*tip
4. *הצג את ערכו של* sum

## יישום האלגוריתם

את ההוראה במבנה *כל עוד... בצע*, ניישם ב-Java במשפט **while**. משפט **while** כולל תנאי בוליאני שקיומו קובע את המשך ביצוע הלולאה.

```
/*
קלט: סדרת מספרים שלמים חיוביים המסתיימת ב-(-1)
פלט: סכומם של המספרים שנקלטו
*/
import java.util.Scanner;
public class SumOfTips
{
```

```
    public static void main (String [] args)
    {
        int tip;          // הטיפ מהמשלוח הנוכחי
        int sum;         // סכום הטיפים המצטבר
        Scanner in = new Scanner(System.in);
1.      sum = 0;
2.      System.out.print("Enter your first tip for today. " +
                                    "End the list of tips with -1: ");
3.      tip = in.nextInt();
4.      while (tip != -1)
        {
4.1.        sum = sum + tip;
4.2.        System.out.print("Enter the next tip. " +
                                     "End the list of tips with -1:");
4.3.        tip = in.nextInt();
        } // while
5.      System.out.println("You have earned " + sum + "shekels");
    }// main
}// SumOfTips
```

## מעקב

נעקוב אחר מהלך ביצוע התוכנית SumOfTips עבור הקלט **12 6 3 -1**:

בטבלת מעקב הכוללת משפט **while** אנו כוללים עמודה עבור התנאי הבוליאני שבכותרת המשפט.

| | **המשפט הבא לביצוע** | tip | sum | tip != -1 | **פלט** |
|---|---|---|---|---|---|
| 1 | sum = 0; | ? | 0 | | |
| 2 | System.out.print("Enter your first tip …"); | ? | 0 | | Enter your first tip … |
| 3 | tip = in.nextInt(); | 12 | 0 | | |
| 4 | **while** (tip!= -1) | 12 | 0 | true | |
| 4.1 | sum = sum + tip; | 12 | 12 | | |
| 4.2 | System.out.print("Enter the next tip. …"); | 12 | 12 | | Enter the next tip. … |
| 4.3 | tip = in.nextInt(); | 6 | 12 | | |
| 4 | **while** (tip != -1) | 6 | 12 | true | |
| 4.1 | sum = sum + tip; | 6 | 18 | | |
| 4.2 | System.out.print("Enter the next tip. …"); | 6 | 18 | | Enter the next tip. … |
| 4.3 | tip = in.nextInt(); | 3 | 18 | | |
| 4 | **while** (tip!= -1) | 3 | 18 | true | |
| 4.1 | sum = sum + tip; | 3 | 21 | | |
| 4.2 | System.out.print("Enter the next tip. …"); | 3 | 21 | | Enter the next tip. … |
| 4.3 | tip = in.nextInt(); | -1 | 21 | | |
| 4 | **while** (tip!= -1) | -1 | 21 | false | |
| 5 | System.out.println("You have earned " + sum + "shekels"); | -1 | 21 | | You have earned 21 Shekels |

***סוף התרגול בשיעה 7***

נציג את המבנה החדש שהכרנו בפתרון לבעיה 7:

**הוראה לביצוע-חוזר-בתנאי** מבטאת ביצוע-חוזר של תת-משימה התלוי בקיום תנאי. הוראה כזאת נכתבת במבנה כלשהו...

גם הוראה כזאת נקראת **לולאה**, והתנאי להמשך הביצוע-החוזר נקרא **תנאי הכניסה ללולאה**.

הוראה לביצוע-חוזר-בתנאי מיושמת ב-Java במשפט **while**.

**המבנה הכללי של משפט while** הוא:

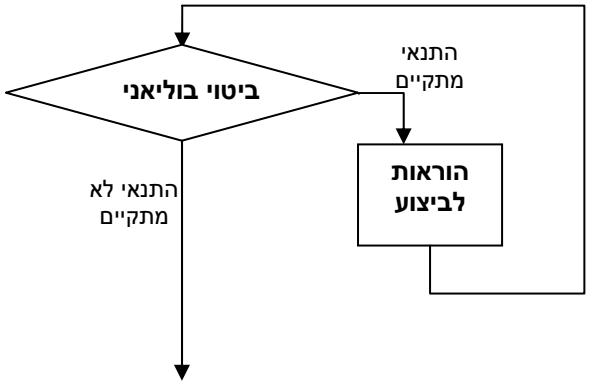
```
while (ביטוי בוליאני)
{
      גוף הלולאה: משפט או משפטים לביצוע
}
```

**ביצוע משפט while** מתחיל בחישוב ערכו של הביטוי הבוליאני. אם ערכו true מתבצע גוף הלולאה. בתום ביצוע גוף הלולאה מחושב הביטוי הבוליאני שוב. אם ערכו true מתבצע גוף הלולאה שוב. תהליך זה נמשך כל עוד ערכו של הביטוי הבוליאני הוא true. כאשר ערכו הוא false מסתיים ביצוע משפט ה-**while**.

**גוף הלולאה** תחום בסוגריים מסולסלים. במקרה שגוף הלולאה מכיל משפט בודד אפשר להשמיט את הסוגריים.

**בטבלת מעקב** אחר מהלך ביצוע תוכנית הכוללת משפט **while** אנו כוללים עמודה עבור תנאי הכניסה ללולאה.

ניתן לתאר משפט **while** באמצעות תרשים הזרימה הבא:




כאמור השוני בין בעיה 7 לבעיות הקודמות לה בפרק הוא בהגדרת הקלט. בבעיות הקודמות ניתן מראש אורך רשימת איברי הקלט, ואילו בבעיה 7 סיום רשימת הקלט מסומן בסימן מיוחד (-1). סימן כזה נקרא **זקיף**.

**זקיף** הוא נתון קלט חריג שתפקידו לסמן את סוף סדרת ערכי הקלט. הזקיף אינו חלק מסדרת הנתונים, ואין לעבד אותו כמו שאר איברי הסדרה.

למשל בפתרון בעיה 7 לא הוספנו את הערך 1- לסכום איברי הסדרה כיוון שהזקיף אינו נחשב כחלק מהסדרה!

השוני בהגדרת הקלט משפיע גם על מבנה הלולאה. הבדל ברור אחד הוא שבפתרון בעיה 7 השתמשנו בהוראה לביצוע-חוזר-בתנאי שיושמה במשפט **while**, בעוד שבפתרון הבעיות

הקודמות השתמשנו בהוראות לביצוע-חוזר מספר פעמים ידוע מראש שיושמו במשפטי **for**. אך יש הבדל נוסף – האופן שמשולבות הוראות הקלט בתוך הלולאה:

בלולאות שבהן השתמשנו בפתרון בעיות קודמות, גוף הלולאה כלל הוראת קלט ומיד לאחריה הוראה לעיבוד הקלט. מבנה זה של גוף הלולאה התאים לבעיות שהיה צריך לעבד בהן כל אחד ואחד מאיברי הקלט באופן אחיד.

בבעיות דוגמת בעיה 7 עיבוד איברי הקלט אינו אחיד: העיבוד של הזקיף שונה מהעיבוד של איברים אחרים בקלט. דרוש מבנה המאפשר לקרוא איבר קלט ולעבד אותו רק אחרי שבוצעה בדיקה כי הוא אינו הזקיף. לכן בפתרון בעיות שהגדרת הקלט בהן כוללת זקיף, מתבצעת הוראת קלט ראשונה לפני הלולאה. גוף הלולאה כולל קודם כל הוראה לעיבוד הקלט, ורק אחר כך הוראת קלט.

המבנה הכללי של הוראה לביצוע-חוזר לעיבוד רשימת קלט המסתיימת בזקיף:

*קלוט ערך*
*כל עוד הערך הנקלט איננו הזקיף בצע:*
*עבד את הערך הנקלט*
*קלוט ערך*

באופן זה, הנתון החדש ייבדק תמיד **מיד** לאחר קליטתו.

---

**שאלה 7.27**

נתון קטע התוכנית הבא. הקלט הוא רשימת תווים המהווים מילה באנגלית ובסופה הזקיף '*'.

```
int i = 0;
char letter;
System.out.print("Enter the first letter of the word: ");
letter = in.next().charAt(0);
while (letter != '*')
{
    i = i + 1;
    System.out.print("Enter the next letter of the word: ");
    letter = in.next().charAt(0);
}
System.out.println(i);
```

מהי מטרת קטע התוכנית?

**שאלה 7.28**

פתחו אלגוריתם אשר הקלט שלו הוא רשימת ציונים (נתונים כמספרים שלמים) בין 0 ל-100 אשר בסופה הזקיף 101, והפלט שלו הוא מספר הציונים ברשימה הגדולים או שווים ל-60. ישמו את האלגוריתם בשפת Java.

**שאלה 7.29**

אלון ובני מתחרים על תפקיד יושב ראש ועדת קישוט של הכיתה, מועמד זוכה אם **יותר** מחצי מן הבוחרים הצביעו עבורו. פתחו וישמו אלגוריתם אשר הקלט שלו הוא סדרה של התווים A ו-B (A עבור אלון, B עבור בני), המבטאת את קולות הבוחרים, ומסתיימת בזקיף '#'. הפלט שלו הוא הודעה אם אלון זכה או לא זכה ברוב קולות (כלומר ביותר מחצי מקולות הבוחרים).

למשל, עבור הקלט #ABBAABBAA הפלט המתאים הוא: Alon wins, ועבור הקלט #ABBABBAA הפלט המתאים הוא: Alon didn't win.

להעמקה בתבנית ***איסוף בקיזוז*** פנו לסעיף התבניות המופיע בסוף הפרק.

**שאלה 7.30**

בבחירות לוועד חיות היער מועמדות שלוש חיות: העכבר שמספרו 1, האריה שמספרו 2, והנמלה שמספרה 3. פתחו אלגוריתם אשר הקלט שלו הוא רשימת קולות הבוחרים (כל קול הוא אחד מן המספרים 1, 2 או 3) ובסופה הזקיף 0. הפלט הוא הודעה המתארת עד כמה צריכות חיות היער להיזהר: אם האריה קיבל פחות מ-30% מהקולות עליהן להיזהר מאוד, אם האריה קיבל בין 31% ל-70% מהקולות עליהן להיזהר קצת, ואם האריה קיבל יותר מ-71% מהקולות הן יכולות להיות רגועות ואינן צריכות להיזהר כלל. ישמו את האלגוריתם בשפת Java.

**שאלה 7.31**

פתחו אלגוריתם אשר הקלט שלו הוא סדרת תווים המהווה מילה באנגלית, ומסתיימת בזקיף '*'. הפלט שלו הוא מילת הקלט והיא מוצפנת באופן הבא: כל אות במילה מוחלפת באות העוקבת לה "בצורה מעגלית" ב-א"ב האנגלי (כלומר, כל אות מלבד האות Z מוחלפת באות העוקבת לה, והאות Z מוחלפת באות A). למשל עבור הקלט *ZEBRA הפלט הוא AFCSB.
ישמו את האלגוריתם בשפת Java.

**שאלה 7.32**

טורניר השש-בש מתחיל, אך איבדתם את הקוביות! פתחו אלגוריתם אשר מדמה הטלת **שתי** קוביות ומציג את תוצאות ההטלה. האלגוריתם מדמה את ההטלות עד שתוצאת ההטלה היא "שש-בש", כלומר צמד המספרים 5 ו-6 או 6 ו-5. לסיום האלגוריתם מציג כפלט את מספר ההטלות שהתבצעו עד שהתקבלה התוצאה שש-בש. ישמו את האלגוריתם בשפת Java.

## ביצוע-חוזר עם תנאי כניסה כלשהו

### פרק 8

**מטרת הבעיה ופתרונה:** הצגת שימוש בהוראה לביצוע-חוזר-בתנאי כניסה כלשהו שאינו תלוי בזקיף.

בתוכנית "הנוסע המתמיד" של חברת התעופה "שחקים" ניתן לצבור מרחקי טיסות. נוסע אשר צובר למעלה מ-3000 קילומטרים זוכה בכרטיס טיסה חינם להונולולו.
פתחו וישמו אלגוריתם אשר הקלט שלו הוא רשימת מרחקי הטיסות של נוסע אשר זכה בכרטיס חינם (כלומר ידוע שכבר צבר יותר מ-3000 ק"מ). הפלט שלו הוא: מספר המרחקים המופיעים ברצף מתחילת הרשימה אשר סכומם המצטבר עולה על 3000, ומספר הקילומטרים שנשארו לנוסע מעבר ל-3000 כְּיִתָרָה לצבירות הבאות.
למשל עבור הקלט: 500 150 700 1000 800 הפלט המתאים הוא 150 5, משום שסכומם של כל חמשת המרחק ים עולה על 3000, ואחרי שמופחת מסכום המרחקים הערך 3000 נשארת יתרה של 150.

## ניתוח הבעיה בעזרת דוגמאות

**שאלה 7.33**

מהו הפלט המתאים עבור כל אחד מן הקלטים הבאים:

א. 200 1000 800 600 600.

ב. 1000 1200 800 900 .

**?** ברור שבפתרון הבעיה יש לצבור את המרחקים הנתונים. כלומר יש לבצע לולאה שבה ייקלט מרחק נתון ויתווסף לצובר. אך כמה פעמים יש לבצע זאת?

בניגוד לבעיות בסעיפים הקודמים שאורך רשימת איברי הקלט בהם היה נתון, או שניתן לנו סימן מיוחד לסיום הקלט (זקיף), בבעיה זו יש לקלוט מרחקים ולהוסיפם לצובר כל עוד ערכו של הצובר איננו גדול מ-3000. ברגע שערכו של הצובר גדול מ-3000 אין צורך להמשיך בפעולות הקליטה והצבירה. מכאן נובע שיש לבצע את פעולות הקליטה והצבירה רק כל עוד מתקיים התנאי הבא:

*המרחק הנצבר קטן או שווה ל-3000*

## פירוק הבעיה לתת-משימות

כל עוד המרחק הנצבר קטן או שווה ל-3000 יש לחזור על ביצוע התת-משימות הבאות:

קליטת מרחק נתון
הוספת המרחק הנתון לצובר
הגדלה ב-1 של "מונה המרחקים"

המונה המוזכר בתת-משימה האחרונה ימנה את מספר המרחקים שנצברו בצובר. כאשר יסתיים ביצוע הלולאה, ישמור מונה זה את מספר המרחקים שנצברו עד שהסכום המצטבר עלה על 3000.

## בחירת משתנים

**dist** – מטיפוס שלם, ישמור מרחק תורן בקלט
**sum** – מטיפוס שלם, צובר שישמור את סכום המרחקים המצטבר
**counter** – מטיפוס שלם, מונה שישמור את מספר המרחקים שנצברו

## האלגוריתם

1. *אתחל את* sum *ב-0*
2. *אתחל את* counter *ב-0*
3. *כל עוד* sum *קטן או שווה ל-3000 בצע:*
   3.1. *קלוט את המרחק ב-*dist
   3.2. *הוסף את* dist *ל-*sum
   3.3. *הגדל ב-1 את* counter
4. *הצג את ערכו של* counter
5. *הצג את ההפרש* 3000-sum

**שאלה 7.34**

א. ישמו את האלגוריתם בשפת Java.

ב. בנו טבלת מעקב אחר ביצוע התוכנית (לפתרון בעיה 8) עבור הקלט:

500 1200 800 300 300

כמה פעמים יתבצע גוף הלולאה?

**שימו ♥:** בכל ביצוע-חוזר של הלולאה נקלט מרחק טיסה אחד, ולכן הערך השמור במשתנה counter הוא בעצם מספר הפעמים שמתבצעת הלולאה.

***סוף פתרון בעיה 8***

**שאלה 7.35**

נסחו תנאי כניסה בוליאני מתאים עבור כל אחד מן התיאורים הבאים של ביצוע-חוזר:

א. סכימת משקלי מכוניות (לצורך מעבר במעבורת) **כל עוד** הסכום אינו עולה על 100 טון.

ב. סכימת המספרים החיוביים השלמים המתחילים ב-1 **עד אשר** הסכום גדול מהערך הנתון כקלט.

ג. קריאת אותיות **עד אשר** נקלטות 10 אותיות A.

**שימו ♥:** בסעיף א מתואר הביצוע-החוזר במתכונת של **כל עוד** ובסעיפים ב ו-ג מתואר הביצוע-החוזר במתכונת של **עד אשר**.

**שאלה 7.36**

בכל אחד מן הסעיפים הבאים מופיע קטע תוכנית הכולל משפט **while** ובו תנאי הכניסה חסר. השלימו את תנאי הכניסה לפי מטרת קטע התוכנית, ובנו טבלת מעקב אחר ביצוע הקטע השלם.

א. מטרת הקטע: הצגה של המספר הזוגי הקטן ביותר אשר גדול מנתון הקלט בהינתן שהקלט הוא מספר חיובי.

```
System.out.print("Enter a number: ");
num = in.nextInt();
i = 0;
while (_____________)
    i = i + 2;
System.out.println(i);
```

בנו טבלת מעקב אחר ביצוע קטע התוכנית השלם עבור הקלט 9.

ב. מטרת הקטע: הצגה של מכפלת שני נתוני הקלט, בהינתן שהם מספרים שלמים חיוביים.

```
System.out.print("Enter a number: ");
x = in.nextInt();
System.out.print("Enter a number: ");
y = in.nextInt();
counter = 0;
sum = 0;
while (_____________)
{
    sum = sum + x;
    counter = counter + 1;
}
System.out.println(sum);
```

בנו טבלת מעקב אחר ביצוע קטע התוכנית השלם עבור הקלט 3 4.

**שאלה 7.37**

נתון קטע התוכנית הבא אשר הקלט שלו הוא מספר חיובי שלם:

```
System.out.print("Enter a number: ");
limit = in.nextInt();
s = 0;
c = 0;
while (s < limit)
{
    c = c + 1;
    s = s + c;
}
System.out.println(c);
```

קטע התוכנית כולל מונה `c` וצובר `s`. הצובר צובר את ערכיו של המונה במהלך הביצוע-החוזר.

א. מהו הפלט עבור הקלט 1? מהו הפלט עבור הקלט 11?

ב. ציינו שני קלטים שונים שעבורם יהיה הפלט 5. מהו מספר הפעמים לביצוע-החוזר עבור קלטים אלה?

ג. מהי מטרת קטע התוכנית?

היעזרו בטבלת מעקב כדי לענות על סעיפים א ו-ב.

**שאלה 7.38**

נתון קטע התוכנית הבא אשר הקלט שלו הוא מספר חיובי שלם, ו-`TOP_LIMIT` הוא קבוע שערכו 100:

```
System.out.print("Enter a number: ");
num = in.nextInt();
mult = 1;
i = 0;
while ((i < num) && (mult < TOP_LIMIT))
{
    i = i + 1;
    mult = mult * i;
}
System.out.println(mult);
```

קטע התוכנית הנתון כולל צובר `mult` אשר צובר ערך של מכפלה, והוא מאותחל בערך 1.

א. מהו הפלט עבור הקלט 2 ? ומהו הפלט עבור הקלט 5?

ב. מהו הקלט שעבורו יהיה הפלט 24? ומהו מספר הפעמים של הביצוע-החוזר עבור קלט זה?

ג. מהי מטרת קטע התוכנית?

היעזרו בטבלת מעקב כדי לענות על סעיפים א ו-ב.

**שאלה 7.39**

פתחו אלגוריתם אשר הקלט שלו הוא מספר חיובי שלם, והפלט שלו הוא החזקה הקטנה ביותר של 2 אשר גדולה מנתון הקלט. למשל: עבור הקלט 7 הפלט הדרוש הוא 8 (כי $2^3=8$), ועבור הקלט 8 הפלט הדרוש הוא 16 (כי $2^4=16$). ישמו את האלגוריתם בשפת Java.

במהלך הפיתוח הקפידו על ניסוח תת-משימות לביצוע-חוזר ועל ניסוח תנאי לביצוע-חוזר.

**שימו ♥:** באלגוריתם זה יש להשתמש בצובר של מכפלה, ולא בפעולת החזקה pow המוגדרת במחלקה Math.

בלולאות **while** שראינו עד כה תנאי הכניסה היו דומים למדי זה לזה. הם כללו תמיד השוואה של משתנה, אשר ערכו גדל במהלך הביצוע-חוזר של הלולאה, לחסם אשר נשמר במשתנה או נקבע מפורשות. הביצוע-החוזר הסתיים כאשר ערכו של המשתנה גדל ועבר את החסם (או השתווה לו). אבל תנאי כניסה כאלה הם רק סוג אחד של תנאי כניסה ללולאה. בפתרון הבעיה הבאה נראה דוגמה לתנאי כניסה מסוג אחר.

## בעיה 9

**מטרת הבעיה ופתרונה:** הצגת תבנית פירוק מספר שלם חיובי כלשהו לספרותיו.

פתחו וישמו אלגוריתם אשר הקלט שלו הוא מספר שלם חיובי והפלט שלו הוא מספר ספרות המספר. למשל: עבור ה קלט 31 הפלט הדרוש הוא 2 ועבור הקלט 15568 הפלט הדרוש הוא 5.

בפרקים קודמים כבר עסקנו בפירוק של מספר שלם לספרותיו, אך שם מספר ספרותיו היה קטן וידוע. בבעיה הנתונה מספר הספרות איננו ידוע ויכול להיות גדול מאוד.

### פירוק הבעיה לתת-משימות

מה יהיו התת-משימות לביצוע-חוזר עבור מניית ספרות המספר? ומה יהיה התנאי לחזרה על ביצוע תת-משימות אלה?

נוכל "לקצץ" את ספרות המספר אחת אחת ולמנות את מספר הספרות הנקצצות. נעשה זאת באמצעות תת-משימות לביצוע-חוזר הכוללות קיצוץ ספרה מן המספר והגדלת מונה ב-1. קיצוץ ספרה יתבצע באמצעות חלוקה בשלמים של המספר ב-10. חלוקה זו תביא לקיצוץ ספרת האחדות (הספרה הימנית ביותר). למשל במקום המספר 534 יתקבל המספר 53.

אם כך, ננסח את התת-משימות הבאות לביצוע-חוזר:

1. קיצוץ ספרת האחדות
2. הגדלה ב-1 של מונה הספרות

יש לבצע תת-משימות אלו כל עוד "יש מה לקצץ", כלומר כל עוד המספר הנותר במהלך הביצוע-החוזר גדול מ-0. לכן התנאי לביצוע-חוזר יהיה: *המספר הנותר גדול מ-0*.

### בחירת משתנים

נבחר שני משתנים מטיפוס שלם:

**num** – ישמור את המספר הניתן כקלט, וספרותיו נקצצות
**digits** – מונה שישמור את מספר הספרות שנקצצו

### יישום האלגוריתם

```
/*
קלט: מספר חיובי שלם
פלט: מספר הספרות במספר הנתון
*/
import java.util.Scanner;
public class DigitCount
{
    public static void main (String [] args)
    {
       int digits = 0; // מספר הספרות
```

```
        int num; // המספר המעובד
        Scanner in = new Scanner(System.in);
        System.out.print("Enter a number: ");
        num = in.nextInt();
        while (num > 0)
        {
            num = num / 10;
            digits++;
        }
        System.out.println("The number of digits is " + digits);
    }// main
}//DigitCount
```

### סוף פתרון בעיה 9

להעמקה בתבניות ***פירוק מספר לספרותיו*** ו***בניית מספר*** פנו לסעיף התבניות המופיע בסוף הפרק.

---

**שאלה 7.40**

א. שנו את התוכנית DigitCount לפתרון בעיה 9, כך שהפלט יכלול את **סכום** ספרות המספר, למשל עבור הקלט 153 הפלט המתאים הוא 9.

ב. שנו את התוכנית DigitCount לפתרון בעיה 9, כך שהפלט יכלול רק את **מספר** הספרות **האי-זוגיות** במספר הנתון. למשל עבור הקלט 150 הפלט המתאים הוא 2.

ג. שנו את התוכנית DigitCount לפתרון בעיה 9, כך שהפלט יכלול רק את **מכפלת** ספרות המספר. שימו לב לאתחול נכון של המשתנה השומר את המכפלה.

**שאלה 7.41**

נתון קטע התוכנית החלקי הבא, אשר הקלט שלו הוא שני מספרים שלמים חיוביים x ו-y ומטרתו היא הצגת שארית החלוקה בשלמים של x ב-y (כלומר תוצאת החישוב x%y). בקטע התוכנית מתבצע החישוב הדרוש באמצעות פעולת **חיסור**.
השלימו את קטע התוכנית.

```
System.out.print("Enter a number: ");
x = in.nextInt();
System.out.print("Enter a number: ");
y = in.nextInt();
while (_____________)
    x = x - y;
System.out.println(_____________);
```

**שאלה 7.42**

שנו את הקטע הנתון בשאלה הקודמת כך שיציג גם את מנת החלוקה של x ב-y, (כלומר את תוצאת החישוב x/y). יש לבצע את החישובים הדרושים באמצעות פעולות חיבור וחיסור בלבד!

**שאלה 7.43**

שני תלמידים המשחקים זה נגד זה מותחים בתחילת המשחק קו באורך N סנטימטרים ($N > 1$). השחקנים מחליפים תורות לסירוגין. כל שחקן מקצר בתורו את הקו לחצי מאורכו. השחקן אשר מקצר בתורו את הקו לאורך של פחות מסנטימטר אחד מנצח במשחק.

למשל אם אורכו של הקו הוא 8 ס״מ, השחקן הראשון יקצר את הקו ל-4 ס״מ והשחקן השני יקצר את הקו ל-2 ס״מ, השחקן הראשון יקצר את הקו לס״מ אחד והשחקן השני יקצר את הקו ל-0.5 ס״מ וינצח במשחק.
פתחו וישמו אלגוריתם אשר הקלט שלו הוא אורכו ההתחלתי של הקו, והפלט שלו הוא הודעה מיהו השחקן המנצח (הראשון או השני).

**שאלה 7.44**

במשחק אסימונים שחקן מניח 2 אסימונים בתור הראשון, 4 אסימונים בתור השני, 8 אסימונים בתור השלישי, וכך הלאה – בכל תור מוכפל מספר האסימונים.
נתון קטע התוכנית הבא אשר הקלט שלו הוא מספר האסימונים ההתחלתי של השחקן, והפלט שלו צריך להיות המספר הסידורי של התור אשר בו לא ניתן להמשיך לשחק לפי השיטה המתוארת. למשל, עבור הקלט המייצג מספר אסימונים התחלתי 9, הפלט הדרוש הוא 3, כיוון שאחרי שהניח 2 אסימונים בתור הראשון ו-4 אסימונים נוספים בתור השני יוותרו לשחקן רק 3 אסימונים (ולא 8 אסימונים, כפי שנדרש לתור השלישי). קטע התוכנית שגוי.

```
int turn = 0;                  //המספר הסידורי של התור הנוכחי
int tokensInCurrentTurn = 0;   //מספר אסימונים למשחק בתור הנוכחי
int totalPlayedTokens = 0;  //סכום אסימונים כולל בתורות שהתבצעו עד כה
int startTokens;               //מספר אסימונים התחלתי
System.out.print("Enter number of tokens to start with: ");
startTokens = in.nextInt();
while (totalPlayedTokens <= startTokens)
{
    turn = turn + 1;
    tokensInCurrentTurn = tokensInCurrentTurn * 2;
    totalPlayedTokens = totalPlayedTokens + tokensInCurrentTurn;
}
System.out.println("The game cannot go on after " + turn + "turns");
```

ציינו מהי השגיאה ותקנו אותה.

**שאלה 7.45**

בצלחת פֶּטְרִי החיידקים מכפילים את עצמם פי 5 בכל דקה עד אשר מספרם עובר סף מסוים הנקרא ״סף ההכפלה״.
פתחו אלגוריתם אשר הקלט שלו הוא שני נתונים: מספר חיידקים התחלתי בצלחת וסף ההכפלה. הפלט שלו הוא מספר החיידקים שיהיו בצלחת בדקה שבה יעבור מספרם את סף ההכפלה. ישמו את האלגוריתם בשפת Java.

---

ראינו עד כה את המבנים ***בזמן מספר פעמים*** (או ***עבור כל... בזמן***)... ו-***כל עוד... בזמן...*** לביצוע-חוזר של תת-משימה. לעתים נוהגים לכנות את המבנה הראשון בשם ״לולאת **for**״ ואת המבנה השני בשם ״לולאת **while**״. מתי נבחר להשתמש במבנה הראשון ומתי נעדיף את המבנה השני?

- כאשר אפשר לחשב לפני ביצוע לולאה את מספר הפעמים שתתבצע, נעדיף לשם הנוחות ולשם הבהירות להשתמש בלולאת **for**.
- כאשר אי אפשר לחשב לפני ביצוע לולאה את מספר הפעמים שתתבצע, בין שהסיום נשלט בידי זקיף ובין שנשלט בידי תנאי אחר, נשתמש בלולאת **while**. המבנה של לולאת **while** מבהיר כי ביצועה תלוי בתנאי, והתנאי עצמו קל לזיהוי.

בחירה נכונה בהוראה לביצוע-חוזר מבהירה לקורא התוכנית אם מספר הפעמים לביצוע ידוע מראש, ואם לא, על פי מה הוא נקבע. לכן בחירה נכונה של הוראה לביצוע-חוזר מסייעת לקריאות ולבהירות התוכנית.

---

**שאלה 7.46**

כתבו לולאה להצגת פלט של 50 כוכביות.

א. כתבו לולאת **for** להצגת הפלט הדרוש.

ב. כתבו לולאת **while** להצגת הפלט הדרוש.

ג. איזה מן הפתרונות פשוט יותר?

**שאלה 7.47**

א. כתבו לולאת **for** המציגה כפלט את 20 הכפולות החיוביות הראשונות של 5 (כלומר הפלט הוא: 5 10 15 ... 100).

ב. כתבו לולאת **for** נוספת המבצעת אותו דבר אבל בעלת כותרת שונה וגוף לולאה שונה.

ג. כתבו לולאת **while** המבצעת אותו דבר.

**שאלה 7.48**

ציינו עבור כל אחת מן הבעיות הבאות אם לפתרונה מתאים יותר להשתמש בלולאת **for** או בלולאת **while**. נמקו את תשובותיכם.

א. הקלט: מספר שלם חיובי num. הפלט: רשימת המספרים השלמים **החיוביים** מ-1 עד num.

ב. הקלט: מספר שלם חיובי num. הפלט: רשימת המספרים השלמים **השליליים** מ-1- עד num-.

ג. הקלט: מספר שלם חיובי num. הפלט: רשימת המספרים השלמים החיוביים מ-1 עד המספר הקטן ביותר k אשר עבורו מתקיים $1+2+3+...+k > num$.

שתי השאלות הבאות מתייחסות להשוואת אותיות באלף-בית האנגלי. נגדיר סדר מילוני של אותיות באופן הבא: אות אחת קטנה מאות אחרת אם האחת מופיעה לפני האחרת בסדר הא"ב. אם האות מופיעה אחרי האחרת בסדר הא"ב אז היא גדולה ממנה. (למשל B קטנה מ-E, ו-F גדולה מ-C).

**שאלה 7.49**

עבור כל אחת מן הבעיות הבאות ציינו אם לפתרונה מתאים יותר להשתמש בלולאת **for** או בלולאת **while**. נמקו בקצרה את תשובותיכם.

**שימו ♥:** בעיות אלו מתאימות לתבניות מקסימום, מינימום, מקום המקסימום ומקום המינימום, אך לא תמיד גודל התחום שמופעלת בו התבנית ידוע מראש.

א. הקלט הוא סדרת אותיות מהאלף-בית ובסופה '*', והפלט הוא האות הגדולה ביותר שמופיעה בקלט.

ב. הקלט הוא מספר שלם חיובי המציין אורך סדרה ואחריו סדרת אותיות מהאלף-בית באורך המצוין, והפלט הוא האות הקטנה ביותר שמופיעה בקלט.

ג. הקלט הוא כמו הקלט לסעיף ב, והפלט הוא המקום הסידורי של ההופעה הראשונה (אולי יש יותר מאחת) של האות הגדולה ביותר שמופיעה בקלט.

ד. הקלט הוא כמו הקלט לסעיף ב, והפלט הוא המקום הסידורי של ההופעה האחרונה (אולי יש יותר מאחת) של האות הקטנה ביותר שמופיעה בקלט.

ה. מהם ההבדלים ביישום האלגוריתם בין סעיף ג ל-ד?

**שאלה 7.50**

פתחו אלגוריתם אשר הקלט שלו הוא סדרת אותיות מן האלף-בית האנגלי שמסתיימת ב-'*', והפלט שלו הוא האות **הגדולה** ביותר מבין האותיות B עד I המופיעות בקלט.

למשל עבור הקלט SCHOOL* הפלט המתאים הוא H. אמנם יש בקלט אותיות גדולות מ-H, למשל S, אך אותיות אלו לא כלולות בסדרת האותיות B עד I.

הניחו שבקלט מופיעה לפחות אות אחת בתחום B עד I.

ישמו את האלגוריתם בשפת Java.

---

## ביצוע-חוזר אינסופי

כאמור, ההוראות לביצוע-חוזר שהוצגו בשלושת הסעיפים הראשונים היו תמיד באורך ידוע מראש. כמובן עבור לולאות כאלו התשובה לשאלה "כמה פעמים יתבצע גוף הלולאה" היא פשוטה מאוד. אבל כאשר מדובר בהוראה לביצוע-חוזר-בתנאי, התשובה לשאלה זו אינה תמיד פשוטה, כפי שמדגימה הבעיה הבאה.

### בעיה 10

**מטרת הבעיה ופתרונה:** דיון בחישוב מספר הפעמים שמתבצעת לולאה, והצגת לולאה אינסופית.

נתונה התוכנית הבאה (מטרתה מתוארת בהערה בתחילת התוכנית):

```
/*
קלט: מספר חיובי שלם
פלט: כל המספרים האי-זוגיים החיוביים הקטנים מן המספר הנתון
*/
import java.util.Scanner;
public class OddNumbers
{
     public static void main (String [] args)
     {
            int limit; // המספר הנתון
            int oddN = 1; //המספר האי-זוגי החיובי הראשון
            Scanner in = new Scanner(System.in);
  1.        System.out.print("Enter a number");
  2.        limit = in.nextInt();
  3.        System.out.println("The odd numbers that are smaller " +
                                        "than the given number are: ");
  4.        while (oddN != limit)
            {
  4.1.         System.out.print(oddN + " ");
  4.2.         oddN = oddN + 2;
            }
     } //main
}// OddNumbers
```

חשבו את מספר הפעמים שתתבצע הלולאה שבתוכנית עבור קלט נתון L.

## ניתוח הבעיה בעזרת דוגמאות

כדי לבטא בצורה כללית (עבור קלט L) את מספר הפעמים שתתבצע לולאת התוכנית נחשב את מספר הפעמים לביצוע הלולאה עבור דוגמאות קלט שונות:

עבור הקלט 1 לא יתבצע גוף הלולאה אפילו פעם אחת, כיוון שערכי `limit` ו-`oddN` שווים כבר בפעם הראשונה שמחושב תנאי הכניסה ללולאה.

עבור הקלט 5, נבחן את מספר הפעמים של ביצוע הלולאה באמצעות טבלת מעקב:

| | **המשפט הבא לביצוע** | `oddN` | `limit` | `oddN!=limit` | **פלט** |
|---|---|---|---|---|---|
| 3 | `System.out.print("The odd Numbers …");` | 1 | 5 | | `The odd Numbers…` |
| 4 | **`while`** `(oddN != limit)` | 1 | 5 | `true` | |
| 4.1 | `System.out.print(oddN + " ");` | 1 | 5 | | 1 |
| 4.2 | `oddN = oddN + 2;` | 3 | 5 | | |
| 4 | **`while`** `(oddN != limit)` | 3 | 5 | `true` | |
| 4.1 | `System.out.print(oddN + " ");` | 3 | 5 | | 3 |
| 4.2 | `oddN = oddN + 2;` | 5 | 5 | | |
| 4 | **`while`** `(oddN != limit)` | 5 | 5 | `false` | |

עבור הקלט 5 יתבצע גוף הלולאה פעמיים ויוצג הפלט: **3 1**.

באופן דומה נוכל לראות שעבור הקלט 15 יתבצע גוף הלולאה 7 פעמים, ויוצג הפלט:

**1 3 5 7 9 11 13**

❓ ננסה להכליל על פי הדוגמאות שבחנו את מספר הפעמים שהלולאה תתבצע עבור קלט אי-זוגי שערכו L. מהו הביטוי הכללי?

עבור קלט אי-זוגי שערכו L הלולאה תתבצע $(L-1)/2$ פעמים.

❓ ניסחנו ביטוי כללי של מספר הפעמים לביצוע הלולאה עבור קלטים אי-זוגיים. האם קיים ביטוי כללי דומה עבור קלטים זוגיים?

נבחן את מספר הפעמים לביצוע הלולאה עבור הקלט 2 באמצעות טבלת מעקב:

| | **המשפט הבא לביצוע** | `oddN` | `limit` | `oddN!=limit` | **פלט** |
|---|---|---|---|---|---|
| 3 | `System.out.print("The odd Numbers …");` | 1 | 2 | | `The odd Numbers…` |
| 4 | **`while`** `(oddN != limit)` | 1 | 2 | `true` | |
| 4.1 | `System.out.print(oddN + " ");` | 1 | 2 | | 1 |
| 4.2 | `oddN = oddN + 2;` | 3 | 2 | | |
| 4 | **`while`** `(oddN != limit)` | 3 | 2 | `true` | |
| 4.1 | `System.out.print(oddN + " ");` | 3 | 2 | | 3 |
| 4.2 | `oddN = oddN + 2;` | 5 | 2 | | |
| 4 | **`while`** `(oddN != limit)` | 5 | 2 | `true` | |
| | | | | | |

קטענו את מילוי הטבלה לפני סיום ביצוע המעקב המלא. ניתן לראות שעבור הקלט 2 מוצג הפלט 3 אשר אין להציגו, כיוון שהוא מספר אי-זוגי שאיננו קטן מן הקלט!

כאשר מחושב הביטוי הבוליאני `oddN != limit` (תנאי הכניסה ללולאה) בפעם השנייה, ערכו של `oddN` הוא 3, וערכו של `limit` הוא 2, ולכן ערכו של הביטוי הבוליאני הוא true ומוצג הערך 3

כפלט. ערכו של `OddN` גדל ב-2 בכל סיבוב בלולאה, ולכן הלולאה תתבצע גם פעם שלישית ויוצג הפלט 5. בעצם, הלולאה תתבצע שוב ושוב וערכו של הביטוי הבוליאני יהיה **תמיד** true, כיוון שערכו של `oddN` רק ילך ויגדל. מכאן נובע שהלולאה תתבצע **אינסוף** פעמים.

הלולאה תתבצע אינסוף פעמים גם עבור הקלט 4 וגם עבור הקלט 6, ובעצם עבור **כל** קלט זוגי. זאת משום שעבור קלט זוגי יתקיים תנאי הכניסה ללולאה שוב ושוב ואף פעם לא יהיה מצב של שוויון של ערכו של `limit` (שהוא מספר זוגי) וערכו של `oddN` (שיהיה תמיד מספר אי-זוגי).

כיוון שעבור קלט זוגי הלולאה תתבצע אינסוף פעמים יוצגו כפלט מספרים אי-זוגיים שאין להציגם, ולכן לא רק שהתוכנית אינה מסתיימת, היא גם מציגה פלט שגוי!

? כיצד ניתן לתקן את התוכנית ולטפל בכך שהלולאה תתבצע מספר מתאים של פעמים גם עבור קלט זוגי?

אפשר לשנות את תנאי הכניסה ללולאה ל-`oddN < limit`, וכך ביצוע הלולאה יסתיים לאחר הצגת המספר האי-זוגי הגדול ביותר שעדיין קטן מהקלט.

---

**שאלה 7.51**

תקנו את התוכנית `OddNumbers` כך שתסתיים לאחר הצגת כל המספרים האי-זוגיים החיוביים שקטנים מן המספר הנתון.

***סוף פתרון בעיה 10***

התוכנית שהוצגה בבעיה 10 כללה לולאה אשר התבצעה כדרוש עבור כל קלט אי-זוגי, אך ביצועה נמשך אינסוף פעמים עבור כל קלט זוגי.

לולאה אשר מתבצעת אינסוף פעמים עבור קלט כלשהו נקראת **לולאה אינסופית**. בחומר הלימוד של "יסודות מדעי המחשב" תוכניות הכוללות לולאה אינסופית נחשבות כתוכניות שגויות.

בפיתוח אלגוריתם קורה לעתים שנכתבת לולאה אינסופית. לולאה כזו עלולה להתבצע אינסוף פעמים רק עבור חלק מן הקלטים. לכן חשוב להקפיד לבדוק לולאת אלגוריתם עבור בחירה ממצה של **דוגמאות קלט מייצגות**, כפי שעשינו בפתרון בעיה 10. בפתרון זה בדקנו תחילה את הלולאה עבור דוגמה של קלט אי-זוגי, וראינו שעבורה מושגת המטרה. אחר כך בדקנו עבור דוגמה של קלט זוגי ונוכחנו שהלולאה אינסופית.

חשבו: מה היה קורה אילו הקלדנו ערך שלילי כקלט לתוכנית?

---

**שאלה 7.52**

ציינו עבור כל אחת מן הלולאות הבאות אם היא לולאה אינסופית.

אם הלולאה סופית ציינו את מספר הפעמים שהיא תתבצע. היעזרו בטבלת מעקב לביצוע החישובים הדרושים.

א.
```
int i = 0;
while (i < 30)
    i = i + 4;
```

ב.
```
int j = 0;
while (j < 50)
    j = j - 10;
```

ג.
```
int j = 0;
while (Math.abs(j) < 30)
    j = j - 10;
```

ד.
```
int k = 0;
for (int j = 1; j < 10; j++)
    k = k + 11;
```

ה. num הוא מספר שלם חיובי כלשהו

```
while (num != 10)
{
    System.out.println(num);
    num = num + 1;
}
```

ו. num הוא מספר שלם חיובי כלשהו

```
while (num != 0)
{
    System.out.println(num);
    num = num - 1;
}
```

**שאלה 7.53**

נתון קטע התוכנית הבא אשר הקלט שלו הוא מספר שלם חיובי:

```
System.out.print("Enter a number: ");
num = in.nextInt();
while (num != 0)
{
    num = num % 3;
    System.out.println(num);
}
```

הלולאה בקטע התוכנית תתבצע מספר סופי של פעמים עבור חלק מן הקלטים, ומספר אינסופי של פעמים עבור שאר הקלטים. לכן הלולאה שבקטע התוכנית היא לולאה אינסופית.

א. תנו דוגמת קלט שעבורה תתבצע הלולאה מספר סופי של פעמים.
מה מאפיין את הקלטים שעבורם תתבצע הלולאה מספר סופי של פעמים?
כמה פעמים תתבצע הלולאה עבור קלטים אלה?

ב. תנו דוגמת קלט שעבורה תתבצע הלולאה מספר אינסופי של פעמים.
מה מאפיין את הקלטים שעבורם תתבצע הלולאה מספר אינסופי של פעמים?

**שאלה 7.54**

נתון קטע התוכנית הבא אשר הקלט שלו הוא שני מספרים שלמים:

```
System.out.print("Enter a number: ");
x = in.nextInt();
System.out.print("Enter a number: ");
y = in.nextInt();
while (x != y)
{
    y = y - 1;
    x = x + 1;
}
System.out.println(x);
```

א. תנו שתי דוגמאות קלט שונות שעבורן **לא יתבצע** גוף הלולאה.

ב. תנו שתי דוגמאות קלט שונות שעבורן יתבצע גוף הלולאה **בדיוק פעם אחת**.

ג. תנו שתי דוגמאות קלט שונות שעבורן יתבצע גוף הלולאה **בדיוק חמש פעמים**.

ד. תנו שתי דוגמאות קלט שונות שעבורן יתבצע גוף הלולאה **אינסוף** פעמים.

ה. נניח שערכו של x הוא 0. עבור אילו ערכים של y יתבצע גוף הלולאה **אינסוף** פעמים?

## 7.5 משתנים מטיפוס בוליאני

בסעיף זה נכיר משתנים מטיפוס בוליאני, כלומר משתנים היכולים לשמור בתוכם אחד משני הערכים **true** או **false**. משתנים כאלה יכולים להשתלב בביטויים בוליאניים ולכן הם שימושיים מאוד בהוראות לביצוע-בתנאי ובהוראות לביצוע-חוזר-בתנאי, כפי שנדגים בהמשך הסעיף.

## בעיה 11

**מטרת הבעיה הבאה:** הצגת משתנה מטיפוס בוליאני

פתחו אלגוריתם שהקלט שלו הוא מספר שלם גדול מ-1 ולאחריו רשימה של מספרים שלמים נוספים הגדולים מ-1. סדרת המספרים מסתיימת במספר 0. אם המספר הראשון שנקרא הוא זוגי, אז הפלט הוא כל המספרים מרשימת המספרים וכל מספר מוצג פעמיים. אם המספר הראשון שנקרא הוא אי-זוגי אז הפלט הוא כל המספרים מרשימת המספרים וכל מספר מוצג פעם אחת בלבד.

למשל עבור הקלט: 0 7 4 2 8 הפלט המתאים הוא 7 7 4 4 2 2.

ועבור הקלט: 0 7 4 2 9 הפלט המתאים הוא 7 4 2.

### פירוק הבעיה לתת-משימות

ננסח רעיון לפירוק ראשוני לתת-משימות לפתרון הבעיה:

1. קליטת מספר ב-num
2. עבור כל מספר ברשימה בדיקה אם num זוגי. אם כן, הצגת המספר הנקלט פעמיים, אחרת הצגת המספר הנקלט פעם אחת

נסתכל על התת-משימה השנייה: כפי שהיא מנוסחת, עבור כל מספר ברשימה אנו בודקים שוב אם num זוגי או לא, אף על פי שבעצם התשובה לשאלה זו תהיה זהה בכל פעם.
לכן, כדאי לבדוק רק פעם אחת אם num זוגי או לא.

הנה פירוק שני לתת-משימות לפתרון הבעיה:

1. קליטת מספר ב-num
2. אם num זוגי
   2.1. עבור כל מספר ברשימה הצגת המספר הנקלט פעמיים
3. אחרת (אם num אי-זוגי)
   3.1. עבור כל מספר ברשימה הצגת המספר הנקלט פעם אחת

אמנם כעת אנו בודקים רק פעם אחת אם num זוגי או לא, אבל אם נסתכל על התת-משימה השנייה והתת-משימה השלישית נראה ששתיהן כוללות קליטה של רשימת המספרים. כלומר, באלגוריתם שהתקבל מהפירוק הזה ולאחר מכן גם בתוכנית ליישום האלגוריתם, הוראות הקלט יופיעו פעמיים.

פירוק בהיר יותר לתת-משימות הוא הפירוק הבא:

1. קליטת מספר
2. בדיקה אם המספר שנקלט זוגי ושמירת תוצאת הבדיקה
   2.1. עבור כל מספר ברשימה:
      2.1.1. אם תוצאת הבדיקה חיובית (כלומר המספר הראשון שנקלט זוגי)
         2.1.1.1. הצגת המספר הנקלט פעמיים
      2.1.2. אחרת (כלומר המספר הראשון שנקלט אי-זוגי)
         2.1.2.1. הצגת המספר הנקלט פעם אחת

בפירוק זה נבדקת הזוגיות של ערך הקלט הראשון רק פעם אחת. גם המעבר על רשימת הערכים מתואר רק פעם אחת. עבור כל ערך קלט ברשימה, מספר המופעים בפלט ייקבע לפי תוצאת הבדיקה שבוצעה קודם לכן.

## בחירת משתנים

מהו טיפוס הערך המתאים לשמירת התוצאה של בדיקת הזוגיות לערך הראשון בקלט? הערכים האפשריים לתשובה לשאלה אם הערך הוא זוגי הם **כן** ו**לא**. טיפוס המתאים לערכים מסוג זה הוא טיפוס **בוליאני**. כזכור מפרקים קודמים, ערך מטיפוס בוליאני יכול להיות אחד משני הערכים: **true** או **false**.

עד כה ראינו ביטויים מטיפוס בוליאני ששימשו אותנו לתיאור תנאים. אבל ניתן גם להצהיר על משתנים מטיפוס בוליאני, ולשלב גם אותם בביטויים בוליאניים. אם נשמור את התוצאה של בדיקת הזוגיות לערך הקלט הראשון במשתנה בוליאני, נוכל להשתמש במשתנה זה כתנאי בהוראה לביצוע-בתנאי שעוברת על רשימת איברי הקלט ומעבדת אותם.

אם כך נשתמש במשתנים הבאים:

**num** – מטיפוס שלם, ישמור את ערך הקלט הראשון
**numInList** – מטיפוס שלם, ישמור את ערך הקלט התורן מהרשימה
**isEven** – מטיפוס בוליאני, ישמור את התוצאה של בדיקת הזוגיות ל-num

## האלגוריתם

מאחר שהרשימה מסתיימת בזקיף, נשתמש בהוראה לביצוע-חוזר-בתנאי התלוי בזקיף.

1. *קלוט מספר* ב-num
2. *בדוק אם* num *זוגי ושמור את תוצאת הבדיקה* ב-isEven
3. *קלוט את המספר הראשון ברשימה* ב-numInList
4. ***כל זמן*** numList ≠ 0 ***בצע:***
   4.1. *אם ערכו של* isEven *הוא* true
      4.1.1. *הצב* *מספיים את* numInList
   4.2. ***אחרת***
      4.2.1. *הצב* *מספר אחת את* numInList
   4.3. *קלוט את המספר הבא ברשימה* ב-numInList

## יישום האלגוריתם

הצהרה על משתנה מטיפוס בוליאני נעשית באמצעות שם הטיפוס **boolean**. לכן ניתן להצהיר על המשתנה `isEven` באופן הבא:

```
boolean isEven;
```

כמו בכל משתנה בתוכנית, גם במשתנים מטיפוס בוליאני אפשר לבצע השמה. הביטוי שבצד ימין של משפט ההשמה צריך להיות ביטוי בוליאני. אנו מעוניינים שהמשתנה `isEven` ישמור את תוצאת בדיקת הזוגיות של ערך הקלט הראשון, ולכן נוכל לכתוב את ההוראה הבאה לביצוע-בתנאי:

```
if (num % 2 == 0)
    isEven = true;
else
    isEven = false;
```

במקרה האחד אנחנו משימים במשתנה `isEven` את הערך הבוליאני **true** ובמקרה האחר את הערך הבוליאני **false**.

אבל מאחר שבצד ימין של משפט השמה למשתנה בוליאני אפשר לכתוב כל ביטוי בוליאני, ולאו דווקא את הביטויים הפשוטים **true** או **false**, נוכל גם לכתוב את משפט ההשמה הבא:

```
isEven = (num % 2 == 0);
```

בעקבות ביצוע ההשמה שלעיל, יושם למשתנה `isEven` הערך **true** אם `num` זוגי, ואחרת יושם בו הערך **false**.

אנו מעוניינים לשלב את המשתנה `isEven` בביטוי הבוליאני בהוראה לביצוע-בתנאי. נוכל לכתוב כך:

```
if (isEven == true)
```

אבל מאחר שערכו של ביטוי בוליאני הוא **true** או **false**, וכך גם ערכו של משתנה בוליאני, הרי משתנה בוליאני הוא כבר ביטוי בוליאני בעצמו (בדיוק כמו שמשתנה שלם הוא כבר ביטוי חשבוני בעצמו). לכן נוכל גם לכתוב:

```
if (isEven)
```

## התוכנית המלאה

```
/*
קלט: מספר שלם ולאחריו רשימת מספרים שלמים גדולים מ-1 שמסתיימת ב-0
פלט: אם המספר הראשון זוגי יוצגו המספרים ברשימה פעמיים, אחרת יוצגו
המספרים ברשימה פעם אחת בלבד
*/
import java.util.Scanner;
public class IfEven
{
    public static void main (String [] args)
    {
        int num;               //המספר הראשון בקלט
        int numInList;         //מספר תורן ברשימת הקלט
        boolean isEven;        //האם המספר הראשון זוגי
        Scanner in = new Scanner(System.in);
        // קליטת המספר הראשון ובדיקה אם הוא זוגי
 1.     System.out.print("Enter the first number: ");
 2.     num = in.nextInt();
 3.     isEven = (num % 2 == 0);
 4.     System.out.print("Enter a number. Enter 0 to end the list:");
 5.     numInList = in.nextInt();
 6.     while (numInList != 0)
        {
 6.1.       if (isEven)
 6.1.1.         System.out.print(numInList + " " + numInList + " ");
 6.2.       else
 6.2.1.         System.out.print(numInList + " ");
 6.3.       System.out.print("Enter the next number. " +
                                        "Enter 0 to end the list");
 6.4.       numInList = in.nextInt();
        } // while
    }// main
}// IfEven
```

**שאלה 7.55**

בנו טבלת מעקב אחר מהלך ביצוע התוכנית `IfEven` לפתרון בעיה 11 עבור הקלט: 0 7 5 4.

***סוף פתרון בעיה 11***

בפתרון הבעיה השתמשנו במשתנה מטיפוס **בוליאני**:

משתנה בוליאני שומר ערכים מטיפוס בוליאני, כלומר אחד משני הערכים **true** או **false**.

במשתנה בוליאני ניתן להשים ערך של ביטוי בוליאני כלשהו.

ניתן לשלב משתנה בוליאני בתוך תנאי בוליאני.

מתאים להשתמש במשתנה בוליאני כדי לזכור תוצאה של בדיקה.

ב-Java משמשת המילה השמורה **boolean** להצהרה על משתנה בוליאני.

בדומה למשתנים מטיפוסים אחרים, ניתן לבצע אתחול למשתנה בוליאני בזמן ההצהרה, למשל:

```
boolean boolVariable = true;
```

למשתנה בוליאני נוהגים לעתים לקרוא **דגל** (flag). כאשר ערכו של המשתנה **true** הוא משול לדגל מורם המסמן מעבר אפשרי. כאשר ערך המשתנה **false** הוא משול לדגל מורד המסמן כי אין אפשרות מעבר.

**שאלה 7.56**

בקטע התוכנית הבא המשתנה `length` שומר מספר חיובי שלם המבטא אורך רשימה. הקטע קולט רשימת תוצאות הטלה של קובייה (תוצאת הטלה של קובייה היא מספר שלם בין 1 ל-6). רשימת התוצאות עלולה לכלול מספרים שגויים (כלומר שאינם בין 1 ל-6). המשתנה `valid` הוא מטיפוס בוליאני ושאר המשתנים הם מטיפוס שלם.

```
i = 1;
s = 0;
valid = true;
while(valid && (i <= length))
{
    System.out.print("Enter the number on the die: ");
    die = in.nextInt();
    if((die >= 1) && (die <= 6))
        s = s + die;
    else
        valid = false;
    i = i + 1;
} // while
if (valid)
    System.out.println(s);
else
    System.out.println(die);
```

א. תארו את הפלט עבור הקלט: 3 2 1 3.

ב. תארו את הפלט עבור הקלט: 3 0 1 3.

ג. מהו תפקיד המשתנה הבוליאני `valid`?

ד. מהי מטרת קטע התוכנית?

**שאלה 7.57**

פתחו אלגוריתם שהקלט שלו הוא רשימה של 20 מספרים חיוביים. הקלט נחשב חוקי אם **כל** המספרים בו הם זוגיים. האלגוריתם בודק אם הקלט חוקי. אם הקלט חוקי האלגוריתם מציג כפלט את סכום המספרים שבקלט, ואחרת תוצג הודעה כי הקלט איננו חוקי. ישמו את האלגוריתם בשפת Java.

**הדרכה**: השתמשו במשתנה בוליאני שיאותחל ב-**true**. אם יימצא בקלט מספר אי-זוגי יושם במשתנה זה הערך **false**.

הבעיה המוצגת בשאלה 7.57 מתאימה לתבנית ***האם כל הערכים בסדרה מקיימים תנאי?*** להעמקה בתבנית פנו לסעיף התבניות המופיע בסוף הפרק.

**שאלה 7.58**

פתחו אלגוריתם שהקלט שלו הוא סיסמה, המורכבת מרשימת אותיות אנגליות המסתיימת ב-'*'. הפלט הוא הודעה המציינת אם הסיסמה תקינה. סיסמה תקינה כוללת לפחות 6 תווים, ובהם לפחות אות אחת קטנה (בתחום a..z) ולפחות אות אחת גדולה (בתחום A..Z). ישמו את האלגוריתם בשפת Java.

**שאלה 7.59**

בקטע התוכנית הבא `length` מכיל מספר שלם חיובי. הקלט לקטע התוכנית הוא סדרה באורך `length` של מילים בנות שלוש אותיות כל אחת.

מטרת קטע התוכנית היא להציג כל מילה בסדרה בסדר אותיות הפוך כל עוד אותיות המילה שונות זו מזו. ברגע שנקלטת מילה שבה לפחות שתי אותיות זהות יש לעצור את מהלך הביצוע.

המשתנה `diff` הוא מטיפוס **בוליאני**, המשתנים `length` ו-`i` מטיפוס שלם והמשתנים `let1`, `let2` ו-`let3` הם מטיפוס תווי.

```
diff = ___________
i = 1;
while(___________________)
{
    System.out.print("Enter the first letter of the word: ");
    let1 = in.next().charAt(0);
    System.out.print("Enter the second letter of the word: ");
    let2 = in.next().charAt(0);
    System.out.print("Enter the third letter of the word: ");
    let3 = in.next().charAt(0);
    if (_________________)
        System.out.println(let3 + " " + let2 + " " + let1);
    else
        diff = _________
} // while
```

השלימו את קטע התוכנית.

## 7.6 הקשר הלוגי לא (not)

הביטויים הבוליאניים שהכרנו עד כה הם ביטויים פשוטים וביטויים מורכבים. הביטויים הבוליאניים הפשוטים שהכרנו כוללים קבועים (**true** או **false**), משתנים בוליאניים, או ביטויי השוואת ערכים בעזרת פעולות השוואה. ביטויים בוליאניים-מורכבים מתקבלים על ידי קישור של ביטויים פשוטים באמצעות קשרים לוגיים. עד כה הכרנו שני קשָרים לוגיים: או (or) ו-וגם (and).

נכיר כעת קשָר לוגי נוסף, הוא הקשר לא (not). כשמו כן הוא – הוא מבטא שלילה, היפוך התנאי. בניגוד לקשרים שהכרנו עד כה, הקשר לא מופעל על ביטוי בוליאני אחד, ולא על שניים.

הנה טבלת האמת של הקשר לא, כאשר b הוא ביטוי בוליאני:

| b | לא b |
|---|---|
| false | true |
| true | false |

סדר העדיפות של הקשר לא גבוה מסדר עדיפותם של הקשרים וגם ו-או. כלומר הוא קודם להם בחישוב ביטוי בוליאני.

בשפת Java הקשר לא נכתב באמצעות סימן קריאה (!).

**דוגמאות**

נניח ש-a הוא משתנה מטיפוס שלם ו-b הוא משתנה מטיפוס בוליאני

- ערכו של הביטוי הבוליאני `true !` הוא **false**.
- ערכו של הביטוי הבוליאני `!(a == 1)` הוא **false** כאשר ערכו של a שווה ל-1, ו-**true** אחרת.
- ערכו של הביטוי הבוליאני `!b` הוא **false** כאשר ערכו של b הוא **true**. כאשר ערכו של b הוא **false**, ערכו של הביטוי `!b` הוא **true**.

---

**שאלה 7.60**

נניח שהמשתנים a ו-b הם מטיפוס שלם. עבור כל ביטוי מן הביטויים הבוליאניים הבאים, תנו דוגמה לערכים למשתנים שעבורם יהיה ערך הביטוי **true**, ודוגמה לערכי המשתנים שעבורם יהיה ערך הביטוי **false**.

| | ערך הביטוי **true** | ערך הביטוי **false** |
|---|---|---|
| א. `!(a != b)` | a = b = | a = b = |
| ב. `!(Math.abs(a) == a)` | a = | a = |

**שאלה 7.61**

נניח שהמשתנים a ו-b הם מטיפוס שלם והמשתנה c הוא מטיפוס בוליאני. עבור כל ביטוי מן הביטויים שבמשפטי ההשמה הבאים, תנו דוגמה לערכי המשתנים שעבורם יושם ב-c הערך **true**, ודוגמה לערכי המשתנים שעבורם יושם ב-c הערך **false**.

| | ערך c הוא **true** | ערך c הוא **false** |
|---|---|---|
| א. `c =(x == Math.sqrt(x))` | `x =` | `x =` |
| ב. `c =(c && (a == b))` | `a =  b =  c =` | `a =  b =  c =` |
| ג. `c =(!((a + b)>= 5))` | `a =  b =` | `a =  b =` |

**שאלה 7.62**

במשחק הניחושים מגריל המחשב מספר בתחום 1-100, והשחקן צריך לנחש אותו. בתום המשחק יודיע המחשב לשחקן כמה ניסיונות ניסה עד שהצליח לנחש.

לפניכם קטע תוכנית ב-Java. השלימו את הקטע כך שיבצע את הנדרש:

```
Random rnd = new Random();
int num = _____________________;       //המספר שהמחשב יגריל
int numOfGuesses = _____________;      //מונה לספירת מספר ניחושים
boolean found = ____________;
while (!found)
{
    numOfGuesses++;
    System.out.print("Enter your guess: ");
    guess = in.nextInt();
    if (_____________)
    {
        System.out.println("Correct!! ");
        found = true;
    }
    else if (_______________)
        System.out.println("Your guess is too big");
    else
        System.out.println("Your guess is too small");
} // while
System.out.println("It took you " + _______________ + " guesses");
```

**שאלה 7.63**

הבעיה עוסקת בהצפנת הודעות. הודעה היא סדרת מילים, כך שכל מילה היא רצף אותיות מן האי"ב האנגלי ובין כל שתי מילים מופיע סימן קריאה (!). הצפנת הודעה מתבצעת באופן הבא: בכל מילה אי-זוגית (כלומר המילה הראשונה, השלישית, החמישית וכו') מוחלפת כל אות באות ה**עוקבת** לה באי"ב. בכל מילה זוגית (כלומר המילה השנייה, הרביעית, השישית וכו') מוחלפת כל אות באות ה**קודמת** לה באי"ב. סימני הקריאה נותרים במקומם ללא שינוי.

כלומר תחילה יש להצפין כל אות נתונה לאות העוקבת לה באי"ב ולאחר כל קריאה של סימן קריאה יש להפוך את "כיוון ההצפנה" (מעוקבת באי"ב לקודמת, או מקודמת באי"ב לעוקבת).

למשל, הצפנת ההודעה DING!DING!DONG! תהיה: EJOH!CHMF!EPOH.

לשם הפשטות נניח שהאותיות A ו-Z אינן נכללות בהודעה.

פתחו וישמו אלגוריתם אשר הקלט שלו הוא מספר המציין אורך של הודעה (כלומר מספר התווים בהודעה) ואחריו ההודעה עצמה (הנקלטת תו אחר תו). הפלט שלו הוא ההודעה כשהיא מוצפנת באופן שתואר לעיל – הצפנת אות תופיע בפלט מיד לאחר קליטתה.

**הדרכה:** השתמשו באלגוריתם במשתנה מטיפוס בוליאני לשמירת "כיוון ההצפנה". אתחלו את ערכו של משתנה זה ל-true, ולאחר כל סימן קריאה הפכו את ערכו באמצעות הקשר ***לא***.

הקשר הבוליאני ***לא*** משמש בתבנית ***האם קיים ערך בסדרה המקיים תנאי?*** להעמקה בתבנית זו פנו לסעיף התבניות המופיע בסוף הפרק.

## 7.7 קינון הוראות לביצוע-חוזר

בהוראה לביצוע-חוזר, גוף הלולאה יכול להכיל הוראות שונות. ייתכן כי בין הוראות אלו יש גם הוראות לביצוע-חוזר. בכך מתקבל מבנה מקונן, הכולל הוראה לביצוע-חוזר בתוך הוראה לביצוע-חוזר. הדבר דומה לקינון של הוראות לביצוע-בתנאי, שהכרנו בפרק 5. בסעיף זה נראה כי מבנה מקונן של הוראות לביצוע-חוזר הוא שימושי מאוד בפתרון בעיות מסוימות.

### *בעיה 12*

**מטרת הבעיה ופתרונה:** הצגת הוראה מקוננת לביצוע-חוזר.

פתחו וישמו אלגוריתם אשר הפלט שלו הוא לוח הכפל, מוצג באופן הבא:

```
1  2  3 ..... 10
2  4  6 ..... 20
3  6  9 ..... 30
.  .  .
.  .  .
10 20 ..... 100
```

בבעיה הנתונה אין קלט. מוגדר פלט בלבד. הפלט הוא טבלת המספרים של לוח הכפל. טבלה זו מכילה 100 מספרים, המסודרים בעשר שורות ובעשרה טורים.

### פירוק הבעיה לתת-משימות

דרך אחת להצגת לוח הכפל היא על ידי אלגוריתם ובו 10 לולאות, אשר כל אחת מהן תציג כפלט שורה של 10 מספרים.
כלומר, נקבל את הפירוק הבא לתת-משימות:

1. הצגה כפלט בשורה אחת של המספרים 1 עד 10
2. הצגה כפלט בשורה אחת של המספרים 1 עד 10, מוכפלים ב-2
3. הצגה כפלט בשורה אחת של המספרים 1 עד 10, מוכפלים ב-3

.
.
.

10. הצגה כפלט בשורה אחת של המספרים 1 עד 10, מוכפלים ב-10

זהו ניסוח מסורבל וארוך. בנוסף ניסוח זה אינו כללי: כדי להציג טבלה בעריכה שונה או טבלה חלקית (למשל רק את חמש השורות הראשונות) יש לכתוב אלגוריתם שונה. בנוסף קשה להכליל את האלגוריתם הזה לבעיה מעט שונה, שגודל הטבלה בה (מספר השורות ומספר העמודות) מתקבל כקלט ואינו ידוע בעת כתיבת האלגוריתם.

האם ניתן לכתוב אלגוריתם פשוט, קצר יותר, ובצורת ניסוח כללית יותר?
כן!
בכל אחת מהלולאות המפורטות מתואר ביצוע-חוזר של תת-משימה. אך בעצם ניתן להתייחס לכל לולאה כאל תת-משימה כוללת יותר, אשר גם על ביצועה יש לחזור 10 פעמים.
כלומר, ניתן לתאר זאת בתת-המשימה הבאה שאותה יש לבצע 10 פעמים:

הצגת השורה הבאה בתור בטבלה

כאשר הצגה של שורה בטבלה אף היא מבוטאת על ידי תת-משימה שיש לבצעה 10 פעמים:

הצגת האיבר הבא בתור בשורה

## בחירת משתנים

מאחר שיש לנו צורך בשתי הוראות לביצוע-חוזר (האחת בתוך האחרת) ואורכן ידוע מראש, נזדקק לשני משתני הבקרה מטיפוס שלם: **i** ו-**j**.

## האלגוריתם

1. עבור כל i שמק מ-1 עד 10 בצע:
   1.1. עבור כל j שמק מ-1 עד 10 בצע:
      1.1.1. הצג כפלט את ערך הביטוי i*j
   1.2. עבור לשורה הבאה

## יישום האלגוריתם

```
/*
התוכנית מציגה כפלט את לוח הכפל של 10*10
*/
public class MultTable
{
    public static void main (String [] args)
    {
        final int TABLE_DIMENSION = 10;
        System.out.println("Mult Table");
1.       for (int i = 1; i <= TABLE_DIMENSION; i++)
        {
1.1.        for (int j = 1; j <= TABLE_DIMENSION; j++)
            {
1.1.1.           System.out.print ((i * j) + " ");
            } // for j
1.2.        System.out.println();// מעבר לשורת הפלט הבאה
        } // for i
    } // main
} // MultTable
```

**שאלה 7.64**

בנו טבלת מעקב אחר מהלך ביצוע התוכנית MultTable.

**סוף פתרון בעיה 12**

במבנה מקונן של לולאות נכללת לולאה פנימית בתוך גוף של לולאה חיצונית.
הלולאה הפנימית מתבצעת כולה בכל סיבוב של גוף הלולאה החיצונית.

בפתרון בעיה 12 הביצוע-החוזר של הלולאה החיצונית כלל 10 ביצועים-חוזרים של גוף הלולאה הפנימית. כיוון שהלולאה החיצונית מתבצעת 10 פעמים, הרי מספר הפעמים הכולל שיתבצע גוף הלולאה הפנימית הוא 100 (10*10=100).

**שימו ♥:** קינון אינו מוגבל רק להוראות לביצוע-חוזר שמספר החזרות בהן ידוע מראש. ההוראה החיצונית יכולה להיות גם הוראה לביצוע-חוזר שתלויה בתנאי, וכך גם ההוראה הפנימית.

בכתיבת מבנה מקונן של לולאות נקפיד על **הזחה**. כיוון שהלולאה הפנימית היא חלק מגוף הלולאה החיצונית, נזיח את הלולאה הפנימית ביחד עם ההוראות של גוף הלולאה החיצונית. הדבר דומה להזחה שבמבנה המקונן של ביצוע-בתנאי.

**שאלה 7.65**

נניח שבאלגוריתם דומה לאלגוריתם המתואר בבעיה 12, משתנה הבקרה של הלולאה החיצונית i יתחיל מהערך 2, ויגדל בכל ביצוע של הלולאה ב-1, עד הגיעו לערך 6. מה יהיה הפלט של אלגוריתם זה? ומה יהיה מספר הפעמים הכולל שיתבצע גוף הלולאה הפנימית שבו?

**שאלה 7.66**

תארו עבור כל אחד מקטעי התוכניות הבאים את פלט קטע התוכנית ואת מספר הפעמים הכולל שמתבצע גוף הלולאה הפנימית:

א.
```
for (int i = 1; i <= 5; i++)
{
    for (int j = 1; j <= 5; j++)
        System.out.print("*");
    System.out.println();
}
```

ב.
```
for (int i = 1; i <= 5; i++)
{
    for (int j = 1; j <= i; j++)
        System.out.print("*");
    System.out.println();
}
```

ג.
```
for (int i = 1; i <= 5; i++)
{
    for (int j = 1; j <= 5; j++)
        System.out.print(j);
    System.out.println();
}
```

ד.
```
for (int i = 1; i <= 5; i++)
{
    for (int j = 5; j >= i; j--)
        System.out.print("*");
    System.out.println();
}
```

**שאלה 7.67**

עבור כל אחד מהסעיפים הבאים פתחו וישמו אלגוריתם אשר הפלט שלו הוא כפי שמופיע בתיאור הסעיף. בכל אחד מהאלגוריתמים מותר לכתוב **רק הוראה אחת** המדפיסה תו או ספרה **והוראה אחת** למעבר שורה בפלט (מכאן נובע כי עליכם להשתמש בלולאות מקוננות):

א.
```
*
**
***
****
*****
```

ב.
```
*****
****
***
**
*
```

ג.
```
*****
****
***
**
*
**
***
****
*****
```

ד.
```
1
12
123
1234
12345
```

ה.
```
1
22
333
4444
55555
```

ו.
```
12345
1234
123
12
1
```

ז.
```
55555
4444
333
22
1
```

ח.
```
11111
2222
333
44
5
```

**שאלה 7.68**

בכיתה 40 תלמידים, כל תלמיד לומד 20 מקצועות. לפניכם תוכנית המחשבת עבור כל תלמיד את ממוצע ציוניו:

```
/*
קלט: 20 הציונים עבור כל אחד מ-40 תלמידי הכיתה
פלט: הממוצע של כל תלמיד ותלמיד
*/
import java.util.Scanner;
public class Grades
{
    public static void main (String [] args)
    {
        final int STUDENT_NUM = 40;
        final int GRADE_NUM = 20;
        double grade;
        double sum;
        double average;
        Scanner in = new Scanner(System.in);
        for(int i = 1; i <= STUDENT_NUM; i++)
        {
            for(int j = 1; j <= GRADE_NUM; j++)
            {
                System.out.print("Enter next student grade: ");
                grade = in.nextInt();
                sum = sum + grade;
            } // for j
        } // for i
    } // main
} // Grades
```

המשפטים הבאים חסרים בתוכנית הנתונה, היכן צריך לשלב את המשפטים האלה כדי שהתוכנית תבצע את הנדרש?

א. `sum = 0;`

ב. `System.out.println(sum / 20);`

**שאלה 7.69**

פתחו אלגוריתם שמקבל כקלט מספר חיובי שלם num, ולאחריו num סדרות באורך לא ידוע של מספרים חיוביים שלמים, כאשר כל סדרה מסתיימת ב-1-. האלגוריתם יציג כפלט את אורכה של הסדרה הארוכה ביותר. ישמו את האלגוריתם בשפת Java.

**שאלה 7.70**

במפעל לייצור נעליים יש עובדים רבים. פתחו אלגוריתם שיקבל כקלט את מספר העובדים במפעל, ואחר כך רשימה של כל המשכורות של השנה האחרונה (12 משכורות) **לכל אחד** מעובדי המפעל. האלגוריתם יחשב וידפיס את **המשכורת האחרונה** של העובד המסכן שסכום משכורותיו כל השנה היה הנמוך ביותר. ישמו את האלגוריתם בשפת Java.

---

בעזרת החומר הנלמד בפרק 7 ניתן לפתור גם בעיות המשתמשות בתבניות ***מציאת כל הערכים בסדרה המקיימים תנאי*** ו***מעבר על זוגות סמוכים בסדרה***. להעמקה בתבניות אלו פנו לסעיף התבניות המופיע בסוף הפרק.

## *סיכום*

בפרק זה למדנו כיצד להורות על ביצוע-חוזר של תת-משימה. הדבר נעשה באמצעות **הוראה לביצוע-חוזר**, והיא הוראת בקרה הנקראת גם **לולאה**.

הכרנו שני מבנים של הוראה לביצוע-חוזר: **הוראה לביצוע-חוזר מספר פעמים ידוע מראש**, ו**הוראה לביצוע-חוזר-בתנאי**.

**הוראה לביצוע-חוזר באורך ידוע מראש** נכתבת בצורה:

*בצע X פעמים:*
*הוראות לביצוע*

או

*עבור כל איבר באוסף ... בצע:*
*הוראות לביצוע*

במבנה הראשון סדרת ההוראות לביצוע מתבצעת X פעמים.

במבנה השני איבריו של התחום המוגדר נסרקים, ועבור כל אחד מהם מבוצעת סדרת ההוראות לביצוע.

**הוראה לביצוע-חוזר-בתנאי** נכתבת בצורה:

*כל עוד <תנאי> בצע:*
*הוראות לביצוע*

התנאי המתואר נקרא **תנאי הכניסה** ללולאה. כל עוד התנאי מתקיים, סדרת ההוראות לביצוע מתבצעת שוב ושוב. כאשר התנאי לא מתקיים, מסתיימת ההוראה לביצוע-החוזר.

בכתיבת הוראה לביצוע-חוזר (מספר פעמים ידוע מראש או בתנאי) אנו מקפידים על **הזחה**: קבוצת ההוראות אשר יש לחזור על ביצוען כתובות כשהן מוזחות פנימה.

קבוצת ההוראות אשר יש לחזור על ביצוען נקראת **גוף הלולאה**.

כאשר מספר הפעמים לביצוע-חוזר אינו ידוע מראש (לפני תחילת ביצוע הלולאה), והוא תלוי בתנאי, נשתמש בהוראה לביצוע-חוזר-בתנאי. בכל מקרה אחר נשתמש בהוראה לביצוע-חוזר באורך ידוע מראש. בכך נסייע ביצירת תוכניות קריאות ובהירות.

בין ההוראות לביצוע-חוזר-בתנאי הבחנו בסוג מסוים: **הוראות לביצוע-חוזר התלויות בזקיף**.

**זקיף** הוא סימן מיוחד המציין את סוף רשימת איברי הקלט. הזקיף **אינו** נחשב כחלק מהקלט. לכן בהוראה לביצוע-חוזר-בתנאי, יש לוודא שאיבר הקלט התורן אינו הזקיף כדי שלא נעבד אותו כחלק מהקלט. זיהוי הזקיף צריך להביא לסיום הביצוע-החוזר.

הוראה לביצוע-חוזר משמשת, בין השאר לביצוע **פעולות צבירה ומנייה**. צבירה נעשית באמצעות צובר, ומנייה באמצעות מונה.

**צובר** הוא משתנה שתפקידו לצבור ערכים (למשל נתונים או תוצאות חישוב). צובר יכול לשמש לפעולות צבירה שונות (למשל, סכום מצטבר או מכפלה מצטברת).

**מונה** הינו משתנה אשר תפקידו למנות אירועים (למשל, מספר המופעים של נתונים). מאחר שהשימוש במונה הוא לצורך ספירה הרי הוא בדרך כלל מטיפוס שלם.

באלגוריתמים שיש בהם שימוש במונה או צובר יש להקפיד עבורם על **אתחול**. ב**אתחול** יושם במונה או בצובר ערך התחלתי המתאים להגדרת תפקידם.

הוראות לביצוע-חוזר משמשות גם לפתרון בעיות שנדרש למצוא בהן **איבר מקסימלי** או **איבר מינימלי** בתוך קבוצת איברים, או ערך נלווה (כמו **מיקומו) של האיבר המקסימלי** (או המינימלי) בקבוצת איברים סדורה.

כאשר אלגוריתם כולל לולאה ניתן לחשב את **מספר הפעמים שהלולאה תתבצע**. מספר זה תלוי בדרך כלל בקלט לאלגוריתם.

יתכן שעבור קלטים מסוימים תתבצע הלולאה אינסוף פעמים. לולאה כזאת נקראת **לולאה אינסופית**. אנו מחשיבים לולאה אינסופית כלולאה שגויה.

בפרק הבא נדגיש את החשיבות של פתרון בעיות באמצעות אלגוריתמים שבהם מספר הפעמים לביצוע-חוזר הוא קטן ככל האפשר.

**משתנה בוליאני** הוא משתנה שיכול לקבל אחד משני הערכים **true** ו-**false**. ניתן להשתמש במשתנים בוליאניים כחלק מביטויים בוליאניים.

**הקשר הלוגי** *לא* (not) משמש ליצירת ביטויים בוליאניים המורכבים מביטויים פשוטים יותר (בדומה לקשרים *וגם* (and) ו-*או* (or)). הוא מופעל על ביטוי בוליאני והוא הופך את ערכו הלוגי (מ-**true** ל-**false**, ולהיפך). עדיפותו של הקשר הלוגי *לא* גבוהה מעדיפותם של הקשרים *וגם* ו-*או*.

הוראה שנמצאת בתוך הוראה לביצוע-חוזר יכולה להיות בעצמה הוראה לביצוע-חוזר. בכך מתקבל **קינון של הוראות לביצוע-חוזר**.

# סיכום מרכיבי שפת Java שנלמדו בפרק 7

**הוראה לביצוע-חוזר מספר פעמים ידוע מראש** מיושמת ב-Java במשפט **for**.

**המבנה הכללי של משפט for** הוא:

```
for (שינוי משתנה הבקרה ; התנאי להמשך הביצוע ;אתחול משתנה הבקרה)
{
    הוראות לביצוע
}
```

כל עוד התנאי להמשך הביצוע (שתלוי בדרך כלל בערכו של משתנה הבקרה) מתקיים, ביצוע הלולאה ממשיך, כלומר מתבצעת סדרת ההוראות לביצוע. סדרת הוראות זו נקראת **גוף הלולאה**.

משתנה הבקרה הוא בדרך כלל מטיפוס שלם. אם אין בו שימוש מעבר לתחום הלולאה ניתן להצהיר עליו בתוך הלולאה, למשל כך:

```
for(int i = 1; i <= 10; i++)
```

**הוראה לביצוע-חוזר-בתנאי** מיושמת ב-Java במשפט **while**.

**המבנה הכללי של משפט while** הוא:

```
while (ביטוי בוליאני)
{
    הוראות לביצוע
}
```

**ביצוע משפט while** מתחיל בחישוב ערכו של הביטוי הבוליאני. אם ערכו **true** מתבצעות ההוראות שבגוף הלולאה. בתום ביצוע גוף הלולאה מחושב ערכו של הביטוי הבוליאני שוב. אם ערכו **true** מתבצעות שוב ההוראות שבגוף הלולאה, וכך הלאה כל עוד ערכו של הביטוי הבוליאני הוא **true**. כאשר ערכו **false** מסתיים ביצוע משפט ה-**while**.

ה**הצהרה על משתנה בוליאני** נעשית בשפת Java באמצעות המילה השמורה **boolean**, למשל כך:

```
boolean flag;
```

**הקשר** נכתב ב-Java באמצעות הסימן !, למשל כך:

```
!(x == 5)
```

# תבניות – פרק 7

פירוט מלא של התבניות ושל שאלות שבפתרונן יש שימוש בתבניות ניתן למצוא באתר הספר ברשת האינטרנט.
בסעיף זה מוצגות תבניות שונות, חלקן מתייחסות לסדרת קלט שאורכה ידוע מראש וחלקן מתייחסות לסדרת קלט שאורכה אינו ידוע מראש.

## מנייה

שם התבנית: **מנייה**
נקודת מוצא: אורך סדרת נתוני הקלט limit, סדרת ערכי הקלט , תנאי מנייה condition
מטרה: מנייה של ערכי הקלט המקיימים את התנאי condition, מתוך סדרת קלט שאורכה limit
אלגוריתם:
1. *אתחל את* count *ב-*0
2. ***כל זמן*** limit ***פעמים:***
   2.1. *קלוט ערך ב-*element
   2.2. ***אם*** element *מקיים את* condition
      2.2.1. *הגדל את* count *ב-*1

שם התבנית: **מנייה**
נקודת מוצא: תנאי סיום conditionToEnd, ערכי הקלט, תנאי מנייה conditionToCount
מטרה: מנייה של ערכי הקלט המקיימים את התנאי conditionToCount. משך ביצוע המנייה תלוי בתנאי conditionToEnd.
אלגוריתם:
1. *אתחל את* count *ב-*0
2. *קלוט ערך ב-*element
3. ***כל זמן לא מתקיים*** conditionToEnd ***כש:***
   3.1. ***אם*** element *מקיים את* conditionToCount
      3.1.1 *הגדל את* count *ב-*1
   3.2. *קלוט ערך ב-*element

## צבירת סכום

שם התבנית: **צבירת סכום**

נקודת מוצא: אורך סדרת נתוני הקלט limit, ערך התחלתי של הצובר initial, ערכי הקלט, תנאי צבירה condition

מטרה: צבירת הסכום של הערך initial ושל ערכי הקלט המקיימים את התנאי condition, מתוך סדרת קלט שאורכה limit

אלגוריתם:

1. *אתחל את* sum *ב-*initial
2. ***בצע*** limit ***פעמים:***
   1. *קלוט ערך ב-*element
   2. ***אם*** element *מקיים את* condition
      1. *הוסף ל-*sum *את ערכו של* element

שם התבנית: **צבירת סכום**

נקודת מוצא: תנאי סיום conditionToEnd, ערך התחלתי של הצובר initial, ערכי הקלט, תנאי צבירה conditionToSum

מטרה: צבירת סכום של הערך initial ושל ערכי הקלט המקיימים את התנאי conditionToSum. משך ביצוע הצבירה תלוי בתנאי conditionToEnd.

אלגוריתם:

1. *אתחל את* sum *ב-*initial
2. *קלוט ערך ב-*element
3. ***כל עוד לא*** *מתקיים* conditionToEnd ***בצע:***
   1. ***אם*** element *מקיים את* conditionToSum
      1. *הוסף ל-*sum *את ערכו של* element
   2. *קלוט ערך ב-*element

## צבירת מכפלה

שם התבנית: **צבירת מכפלה**

נקודת מוצא: אורך סדרת נתוני הקלט limit, ערך התחלתי של הצובר initial, ערכי הקלט, תנאי צבירה condition

מטרה: צבירת מכפלה של הערך initial ושל ערכי הקלט המקיימים את התנאי condition, מתוך סדרת קלט שאורכה limit

אלגוריתם:

1. *אתחל את* mult *ב-*initial
2. ***בצע*** limit ***פעמים:***
   1. *קלוט ערך ב-*element
   2. ***אם*** element *מקיים את* condition
      1. *הכפל את* mult *ב-*element ***והשם את המכפלה ב-***mult

שם התבנית: **צבירת מכפלה**

נקודת מוצא: תנאי סיום conditionToEnd, ערך תחילי של הצובר initial, ערכי הקלט, תנאי צבירה conditionToMult

מטרה: צבירת מכפלה של הערך initial ושל ערכי הקלט המקיימים את התנאי conditionToMult. משך ביצוע הצבירה תלוי בתנאי conditionToEnd.

אלגוריתם:

1. אתחל את mult ב-initial
2. קלוט ערך ב-element
3. כל עוד לא מתקיים conditionToEnd בצע:
   3.1. אם element מקיים את conditionToMult
      3.1.1. הכפל את mult ב-element והשם את המכפלה ב-mult
   3.2. קלוט ערך ב-element

## ממוצע של סדרת מספרים

שם התבנית: **ממוצע של סדרת מספרים**

נקודת מוצא: תנאי סיום conditionToEnd, ערכי הקלט, תנאי conditionToInclude

מטרה: חישוב ממוצע של ערכי הקלט המקיימים את התנאי conditionToInclude. ביצוע החישוב תלוי בתנאי conditionToEnd.

אלגוריתם:

1. אתחל את count ב-0
2. אתחל את sum ב-0
3. קלוט ערך ב-element
4. כל עוד לא מתקיים conditionToEnd בצע:
   4.1. אם element מקיים את conditionToInclude
      4.1.1. הגדל את count ב-1
      4.1.2. הוסף לסכום שלו sum את element
   4.2. קלוט ערך ב-element
5. השם ב-average את סכום הביטוי הרשום sum/count

## מציאת מקסימום או מינימום בסדרה

בשל הדמיון האלגוריתמי בין מציאת מקסימום למציאת מינימום וכדי לא לחזור בפירוט על התבניות, נפרט את התבניות באופן הבא: עבור התבנית ***מציאת מקסימום*** נתייחס לסדרה שאורכה ידוע מראש ואילו עבור התבנית ***מציאת מינימום*** נתייחס לסדרה שאורכה אינו ידוע מראש.

שם התבנית: **מציאת מקסימום בסדרה**

נקודת מוצא: אורך סדרת נתוני הקלט limit, ערכי הקלט

מטרה: מציאת הערך הגדול ביותר בסדרת ערכי הקלט שאורכה הוא limit

אלגוריתם:

1. *קלוט ערך ב-*max
2. ***בצע*** limit-1 ***פעמים:***
   2.1. *קלוט ערך ב-*element
   2.2 ***אם*** element > max
      2.2.1 *השם ב-*max *את הערך של* element

שם התבנית: **מציאת מינימום בסדרה**

נקודת מוצא: תנאי סיום conditionToEndMinSearch, ערכי הקלט

מטרה: מציאת הערך הקטן ביותר מבין ערכי הקלט, עד אשר מתקיים התנאי conditionToEndMinSearch

אלגוריתם:

1. *קלוט ערך ב-*min
2. *קלוט ערך ב-*element
3. ***כל עוד לא מתקיים*** conditionToEndMinSearch ***בצע:***
   3.1. ***אם*** element < min
      3.1.1. *השם ב-*min *את הערך של* element
   3.2. *קלוט ערך ב-*element

## מציאת ערך נלווה למקסימום או למינימום בסדרה

גם כאן בשל הדמיון האלגוריתמי בין התבניות, נפרט את התבניות באופן הבא: עבור התבנית ***מציאת ערך נלווה למקסימום בסדרה*** נתייחס לסדרה שאורכה ידוע מראש ואילו עבור התבנית ***מציאת ערך נלווה למינימום בסדרה*** נתייחס לסדרה שאורכה אינו ידוע מראש. בתבניות הבאות נמצא מיקום של איבר כערך נלווה.

שם התבנית: **מציאת ערך נלווה למקסימום בסדרה**
נקודת מוצא: אורך סדרת נתוני הקלט limit, ערכי הקלט
מטרה: מציאת מיקום הערך הגדול ביותר בסדרת ערכי הקלט שאורכה הוא limit
אלגוריתם:
1. *קלוט ערך ב-*max
2. *אתחל את* placeOfMax *ב-*1
3. ***בצע*** *1-*limit ***פעמים:***
   3.1. *קלוט ערך ב-*element
   3.2. ***אם*** element > max
      3.2.1. *השם ב-*max *את הערך של* element
      3.2.2. *השם ב-*placeOfMax *את מיקומו של* element ***בקלט***

שם התבנית: **מציאת ערך נלווה למינימום בסדרה**
נקודת מוצא: תנאי סיום conditionToEnd, ערכי הקלט
מטרה: מציאת מיקום הערך הקטן ביותר מבין ערכי הקלט, עד אשר מתקיים התנאי conditionTtoEnd
אלגוריתם:
1. *קלוט ערך ב-*min
2. *אתחל את* placeOfMin *ב-*1
3. *קלוט ערך ב-*element
4. *אתחל את* currentPlace *ב-*2
5. ***כל עוד לא מתקיים*** conditionToEnd ***בצע:***
   5.1. ***אם*** element < min
      5.1.1. *השם ב-*min *את הערך של* element
      5.1.2. *השם ב-*placeOfMin *את הערך של* currentPlace
   5.2. *הגדל את* currentPlace *ב-*1
   5.3. *קלוט ערך ב-*element

## איסוף בקיזוז

שם התבנית: **איסוף בקיזוז באמצעות מנייה**

נקודת מוצא: אורך סדרת נתוני הקלט limit, ערכי הקלט, תנאי לקיזוז conditionToDecrease

מטרה: איסוף בקיזוז באמצעות מנייה של ערכי הקלט מתוך סדרה שאורכה limit. הקיזוז מתבצע על פי התנאי conditionToDecrease

אלגוריתם:

1. אתחל את count ב-0
2. בזמן limit עשה:
   1.1. קלוט ערך ב-element
   2.2. אם element לא מקיים את conditionToDecrease
      2.2.1. הגדל את count ב-1
   2.3. אחרת
      2.3.1. הקטן את count ב-1

שם התבנית: **איסוף בקיזוז באמצעות צבירה**

נקודת מוצא: תנאי סיום conditionToEnd, ערכי הקלט, ערך צבירה התחלתי initial, תנאי לקיזוז conditionToDecrease

מטרה: איסוף בקיזוז באמצעות צבירת סכום של ערכי הקלט ושל הערך initial. משך הביצוע תלוי בתנאי conditionToEnd, והקיזוז מתבצע על פי התנאי conditionToDecrease

אלגוריתם:

1. אתחל את sum ב-initial
2. קלוט ערך ב-element
3. כל עוד לא מתקיים conditionToEnd בצע:
   3.1. אם element לא מקיים את conditionToDecrease
      3.1.1. הוסף ל-sum את ערכו של element
   3.2. אחרת
      3.2.1. הפחת מ-sum את ערכו של element
   3.3. קלוט ערך ב-element

## פירוק מספר חיובי לספרותיו

שם התבנית: **פירוק מספר חיובי לספרותיו**

נקודת מוצא: מספר שלם חיובי num

מטרה: הצגה כפלט של ספרותיו של num

אלגוריתם:

1. כל עוד num שונה מ-0 בצע
2. הצג כפלט את ספרת האחדות של num
3. הקטן את num פי 10

## בניית מספר

שם התבנית: **בניית מספר**
נקודת מוצא: אורך סדרת נתוני הקלט limit, ספרות הקלט
מטרה: בניית מספר מספרות הקלט
אלגוריתם:
1. *אתחל את* num *ב-*0
2. ***כל עוד*** limit ***פעמים:***
   2.1. *קלוט ספרה ב-*digit
   2.2. *השם ב-*num *את הערך שלו הביטוי הבא:* num * 10 + digit

## האם כל הערכים בסדרה מקיימים תנאי?

שם התבנית: **האם כל הערכים בסדרה מקיימים תנאי?**
נקודת מוצא: אורך סדרת נתוני הקלט limit, ערכי הקלט, תנאי condition
מטרה: קביעת הערך true אם כל הערכים בסדרה מקיימים את התנאי condition וקביעת הערך false אם קיים ערך אחד בסדרה שאינו מקיים את התנאי
אלגוריתם:
1. *אתחל את* all *ב-*true
2. *אתחל את* howmany *ב-*1
3. ***כל עוד*** howmany ≤ limit ***ו***-*אם ערכו של* all *הוא* true ***בצע:***
   3.1. *קלוט ערך ב-*element
   3.2. *הגדל את ערכו של* howMany *ב-*1
   3.3. ***אם*** element *אינו מקיים את* condition
      3.3.1. *השם ב-*all *את הערך* false

## האם קיים ערך בסדרה המקיים תנאי?

שם התבנית: **האם קיים ערך בסדרה המקיים תנאי?**
נקודת מוצא: תנאי לסיום הסדרה toEnd, ערכי הקלט, תנאי condition
מטרה: קביעת הערך true אם קיים ערך בסדרה המקיים את התנאי condition וקביעת הערך false אם כל הערכים בסדרה אינם מקיימים את התנאי
אלגוריתם:
1. *אתחל את* found *ל-*false
2. *קלוט ערך ב-*element
3. ***כל עוד*** *התנאי* toEnd *לא מתקיים* ***ו***-*אם ערכו של* found *הוא* false ***בצע:***
   3.1. ***אם*** element *מקיים את* condition
      3.1.1. *השם ב-*found *את הערך* true
   3.2. ***אחרת***
      3.2.1. *קלוט ערך ב-*element

## מציאת כל הערכים בסדרה המקיימים תנאי

שם התבנית: **מציאת כל הערכים בסדרה המקיימים תנאי**
נקודת מוצא: תנאי לסיום הסדרה toEnd, ערכי הקלט, תנאי condition
מטרה: הצגה כפלט של כל ערכי הקלט המקיימים את התנאי condition
אלגוריתם:

1. קלוט ערך ב-element
2. כל עוד התנאי toEnd לא מתקיים בצע:
   3.1. אם element מקיים את condition
      3.1.1. הצג כפלט את ערכו של element
   3.2. קלוט ערך ב-element

## מעבר על זוגות סמוכים בסדרה

שם התבנית: **מעבר על זוגות סמוכים בסדרה**
נקודת מוצא: אורך סדרת נתוני הקלט limit, ערכי הקלט
מטרה: הצגה כפלט של סכומי כל זוגות הערכים הסמוכים בסדרת הקלט שאורכה limit
אלגוריתם:

1. קלוט ערך ב-beforeLast
2. בצע limit-1 פעמים
   2.1. קלוט ערך ב-last
   2.2. הצג כפלט את הערך שהביטוי הבא beforeLast + last
   2.3. השם ב-beforeLast את הערך של last